UNIVERSITY OF ALBERTA

COLLISION-FREE PATH PLANNING IN THE THREE-DIMENSIONAL WORK SPACE OF A REVOLUTE ROBOT

BY

OLUWATOLAMISE ADEODU

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH IN PARTIAL FUFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF MECHANICAL ENGINEERING

EDMONTON, ALBERTA

Fall 1994

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-94996-1

Canada

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Oluwatolamise Adeodu

TITLE OF THESIS: Collision-Free Path Planning In The Three-Dimensional Work Space Of A Revolute Robot

DEGREE: Master of Science

YEAR THIS DEGREE GRANTED: 1994

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

204 Michener Park
Edmonton, Alberta
Canada
T6H 4M5

Date: Sept 6/94

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled ***Collision-Free Path Planning In The Three-Dimensional Work Space Of A Revolute Robot*** submitted by ***Oluwatolamise Adeodu*** in partial fulfillment of the requirements for the degree of Master of Science.

Roger W Toogood

Supervisor: Dr. R.W. Toogood

Gary Faulkner

Dr. M.G. Faulkner

Q-H Meng

Dr. Q-H. M. Meng

Date: 2 Sept 94

EBENEZER

The men of Israel went out of Mizpeh and pursued the Philistines and smote them until they came to a point below Bethcar. Then Samuel took a stone and set it between Mizpeh and Shen and called it Ebenezer saying, "Thus far has the Lord helped us".

PATH PLANNING PAR EXCELLENCE

Trust in the Lord with all your heart and do not lean on your own understanding. In all your ways acknowledge Him and He will direct your paths.

1 Sam.7: 11-12 & Pro.3: 5-6

The Book

ABSTRACT

This thesis presents an obstacle avoidance program (OBSTAP) written for the Mitsubishi RM 101 revolute robot. For a given robot task which causes collisions with stationary obstacles placed anywhere in the robot's three-dimensional work space, OBSTAP quickly detects the collisions and heuristically plans a path around the obstacles. In the program, a robot link is shrunk to a line while each obstacle is grown and described in world space. OBSTAP allows for moderate cluttering of the work space and collision is avoided for the gripper and all the links of the robot.

ACKNOWLEDGEMENT

I am indebted to several people for their help on this work. My thanks go to Dr. R.W. Toogood for his patient supervision of the research. He furnished me with careful guidance especially at the crucial stage of problem formulation. I am also grateful to the technical staff of the Department of Mechanical Engineering for their assistance when I got to testing OBSTAP.

I greatly appreciate the prayerful support of my wife Nike and of my children Seyi, Ibukun and Kemi. They sustained me wonderfully especially at those times when the program refused to work because of one bug or the other.

Many thanks to you all!

TABLE OF CONTENTS

CHAPTER 1 LITERATURE REVIEW AND THESIS OUTLINE 1

1.1 Literature Review 1

1.1.1 World Modelling 2

1.1.2 Programming Methods 5

1.1.3 Potential Field Methods 6

1.1.4 Work Space Mapping 7

1.1.5 Sensing Devices 9

1.1.6 Path Optimisation 10

1.2 Thesis Outline 12

CHAPTER 2 SCOPE OF OBSTAP 14

2.1 The Mitsubishi RM 101 Robot 14

2.2 Uses of OBSTAP 17

CHAPTER 3 OBSTACLE DETECTION 21

3.1 Geometric Tools 21

3.2 Growth of Obstacles 23

3.3 Instantaneous Robot Position 25

3.4 Collision Checks 27

3.5 Detection Algorithm 33

CHAPTER 4 PATH PLANNING 36

4.1 Heuristic Search 36

4.2 First Level Path 37

4.2.1 Cases of Variable Body Joint Angle . . 40

4.2.1.1 Collision With a Vertical Near Side 40

4.2.1.2 Collision With Top or Bottom or Vertical Far Side 44

4.2.2 Cases of Fixed Body Joint Angle 46

4.2.2.1 Gripper Initially Moving Down . 48

4.2.2.2 Gripper Initially Moving Up . . 48

4.2.3 Other Factors For First Level 51

4.3 Second Level Path 51

CHAPTER 5 TEST RESULTS AND DISCUSSION 55
5.1 Description of Test Obstacles 55
5.2 Description of Test Tasks 62
5.3 Test Results 62
5.4 Discussion 71
5.5 Recommendation On Safety. 78
5.6 Application of OBSTAP to Other Robots 80
5.7 Limitations of the Heuristic Search 81

CHAPTER 6 CONCLUSION 82

REFERENCES 85

APPENDIX: OBSTAP USER'S MANUAL 92
A1. Uses of OBSTAP 92
A2. Description of Robot Task 92
A3. Description of Obstacles 94
A4. Obstacle Detection Only 96
A5. Obstacle Detection With Path Planning 97
A6. Robot Safety and Path Animation 98

LIST OF TABLES

Table 2.1: Denavit-Hartenburg Parameters For The RM 101 16

Table 2.2: RM 101 Joint Angle Limits and Step Sizes 16

Table 5.1(a): Dimensions and Locations of Ungrown and Unconnected Obstacles 59

Table 5.1(b): Dimensions and Locations of Ungrown Connected Obstacles (First Set) 60

Table 5.1(c): Dimensions and Locations of Ungrown Connected Obstacles (Second Set) 61

Table 5.2: Data For Robot Task Files 64

Table 5.3: Test Results (1) 65

Table 5.4: Test Results (2) 66

Table 5.5: Test Results (3) 67

LIST OF FIGURES

Fig. 1.1: Shrinking Gripper and Payload and Growing Obstacles 3
Fig. 2.1: Coordinate Frames for the RM 101 Robot . . . 15
Fig. 2.2: Outlines and Dimensions of the RM 101 Robot 15
Fig. 3.1: Stretching a Bar Equally at Both Ends 24
Fig. 3.2: Growing an Obstacle Represented as a Cuboid 24
Fig. 3.3: Checking for Collision Between Two Line Models 29
Fig. 3.4: Checking for Collision Between (i) A Point and a Plane and (ii) A Line and a Plane 31
Fig. 3.5: Obstacle Detection Flow Chart 32
Fig. 4.1: Path Planning Flow Chart 39
Fig. 4.2: Collision With a Vertical Near Side of an Obstacle (Robot Body Moving) 41
Fig. 4.3: Collision With the Top or Bottom or a Vertical Far Side of an Obstacle (Robot Body Moving) 45
Fig. 4.4: Collision With Gripper Moving Down (Fixed Body Joint Angle) 47
Fig. 4.5: Collision With Gripper Moving Up (Fixed Body Joint Angle) 49
Fig. 5.1: Two Views of Four Unconnected Obstacles (File OB4.DAT) 56
Fig. 5.2: Two Views of Six Connected Obstacles (Set #1, File COB1.DAT) 57
Fig. 5.3: Two Views of Five Connected Obstacles (Set #2, File COB2.DAT) 58
Fig. 5.4: Test #8 Showing Original Motor File (TEST12.MOT) and Two Unconnected Obstacles (OB2.DAT) 68

Fig. 5.5: Test #8 Showing the Planned First Level Path Around Obstacles OB2.DAT 69
Fig. 5.6: Test #8 Showing the Planned Second Level Path Around Obstacles OB2.DAT 70
Fig. A1: Obstacle Description in OBSTAP 95

CHAPTER 1

LITERATURE REVIEW AND THESIS OUTLINE

This chapter gives an overview of path planning in robotics. An introduction to the research described in the thesis is also given. Reference numbers are written in square brackets.

1.1 Literature Review

Many accidents have occurred in the application of robots [1 - 4]. The accidents include collisions with obstacles or other robots and injury to personnel. Suggestions as to how accidents could be prevented include ensuring that nothing stands in the way of a moving robot. This recommendation has been made especially for robots that have ill defined or complex work spaces [5]. The revolute robot falls into this category.

However, of necessity, there may be an obstacle in the work space of a robot. Such an obstacle could be a support structure or some other robot with which the first has to cooperate. Rectangular manipulators have well defined work spaces and there has been a lot of success in obstacle avoidance with them. The same is not true for the revolute type [6, p 411]. For revolute manipulators, the bigger part of research work in obstacle avoidance has concentrated on the end-effector and payload combination in a two-dimensional space.

In the following sections, we shall focus on gross motions which involve large displacements of robot links. Fine motions are not considered in this thesis.

Many obstacle avoidance methods have been tried. These methods and their scope will now be discussed. Some of the references listed report on path planning for fixed robots

(manipulators) while other references are for mobile robots. Furthermore, some of the methods only simulate robotic systems and have not been tested experimentally.

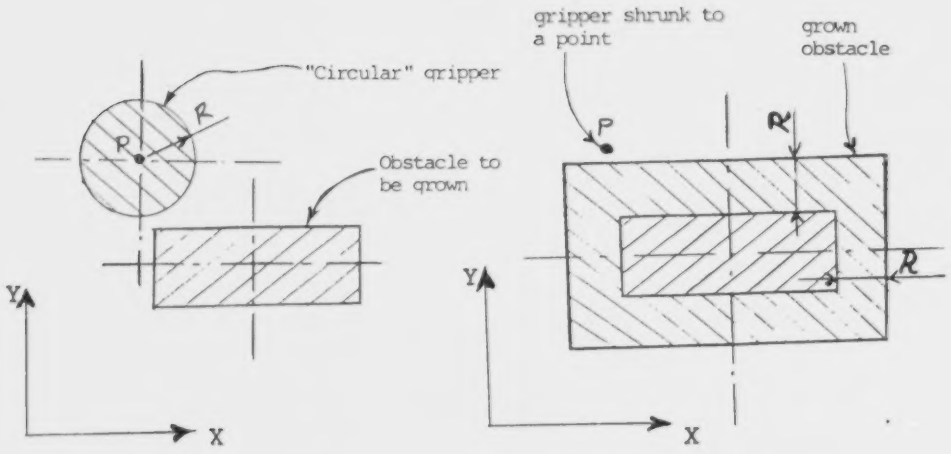
1.1.1 World Modelling

The gripper of a robot can collide with an obstacle in an infinite number of ways. Thus checking for collisions by using very small elements of the gripper or obstacle would be computationally intensive.

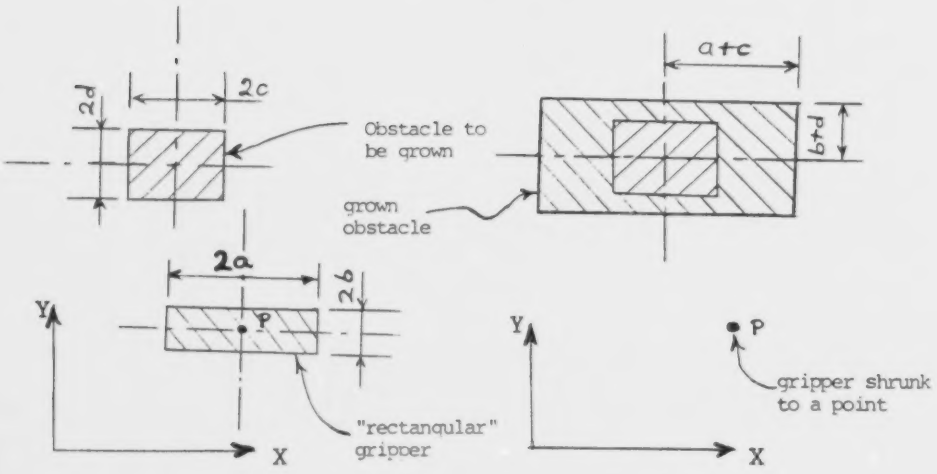
The use of simplifying world models to describe objects in the work space has received much attention in robotics. Simple lines, surfaces or primitive solids are used to model the actual physical system. For example, polygons, polyhedra and generalized cones have been used to model objects [7 - 9].

A list of edge descriptions for an object has also been used. This is called the wire frame method. A problem with edge descriptions, however, is that the description is not unique. Two different objects may have the same wire frame description. Markowsky and Wesley [10] have suggested that a list of edge descriptions cannot adequately describe a unique polyhedron.

Also, to reduce the amount of computation, gripper geometry has been simplified by shrinking it to a point. Correspondingly obstacles have been grown [11]. This process is illustrated in two dimensions in Fig. 1.1 (a,b). In (a) the gripper is initially modelled as a circle while in (b) it is initially modelled as a rectangle. Relative rotation between the gripper and obstacle can be permitted in case (a) because the amount by which the obstacle is grown is independent of the orientation of the gripper. Initially the method illustrated in (a) may be considered as being useful for revolute manipulators.



(a) "Circular" Gripper Model.



(b) "Rectangular" Gripper Model.

Fig. 1.1: Shrinking Gripper and Payload and Growing Obstacles.

However, modelling with circles and spheres is not recommended for this class of robots. The use of circles in two dimensions and spheres in three for object representation does not efficiently represent the available work space. A reason for this is that only one parameter (a radius) is used in the process of growing the obstacle. There is no flexibility regarding the shape and orientation of the grown obstacle relative to the world axes. Relative rotation is not allowed in case (b). This has been found to have good application with rectangular manipulators.

Apart from using world space, obstacles have also been modelled using what is termed configuration space [12 - 16]. In programs written using the c-space approach, obstacles are described in terms of varying robot parameters (for example, link lengths for rectangular robots and joint angles for revolute robots). The method is essentially a mapping of the surface of an obstacle to a space whose axes are given by the robot parameters. The fewer the number of parameters, the simpler the mapped shape is. For example, for a 2 - R robot the shape obtained for a three-dimensional obstacle will be like an area. Only points outside of the contour which defines the area are collision-free. The shape obtained will seem to have volume for a 3 - R robot. Only points outside of the volume are collision-free. Clearly the shape is very difficult to visualise for practical manipulators which have up to six revolute joints.

The use of c-space has been more successful with rectangular robots than for revolute. This is because an obstacle can be modelled in term of linear dimensions (length, width and height) just as changes in the configurations of a rectangular robot are in terms of corresponding linear dimensions. For collisions involving the payload and gripper alone, the c-space approach yields a gripper path that is tangential to the obstacle profile in the vicinity of the obstacle.

1.1.2 Programming Methods

Three robot programming methods have been developed [6]. They are:

(a) on-line teaching of the robot,
(b) off-line robot programming using a robot-level computer language,
(c) off-line task-level robot programming.

On-line teaching is the oldest of the methods. By manually leading the robot through the desired path, the robot is taught its path for a task (i.e the moves to be made) and the functions to be performed at specific points along the path. Using this method, it is easy to make the robot avoid obstacles such as nearby fixed structures. This is termed continuous path control. Teach pendants are also used for point-to-point control. The knot points (or intermediate points) in the path are generally widely spaced for point-to-point control and closely spaced for continuous path control. These knot points are then stored in memory. There are memory limitations in the case of continuous path control. To perform the task, the robot then has to play back the stored information.

On-line teaching is time-consuming. Also, since the robot is programmed in its work cell, valuable production time is lost during the process. Furthermore, it does not facilitate the use of a robot for a variety of tasks which it is easily capable of doing. Further details on robot teaching can be found in [17].

Off-line robot programming does not require that the programmer be near the robot to physically manipulate its controls. By modelling the system at a remote station, he can compiles files which can be downloaded to the robot at a later stage. Off-line programming therefore has the advantage that it does not cause the loss of valuable production time. Many computer languages have been used to program robots. Some of

these languages have already been applied in industry. Lozano-Perez [18] and Gruver et al. [19] have given good overviews of robot-level computer languages. A good discussion on off-line programming has also been given by Yong et al. [20]. Developments suggest that BASIC and C are getting ahead of the other languages used for robotic software [21, p441].

Robot-level programming has the advantage that a robot can be easily reprogrammed to carry out other tasks. Another advantage is that it is possible for the robot to interact with any sensor in use. However, it has the disadvantage that the level of expertise required for the programming is high. Such expertise is not usually available on a typical shop floor and acquiring it may be expensive. As high expertise is not required in the case of on-line teaching.

The third method (task-level programming) can be distinguished from the second by its more descriptive, less mathematical nature. It describes the effect of a robot action on objects placed in the work volume of the robot. Such a program may have statements which stipulate that the robot PICK a block, MOVE a specified distance and PLACE the block relative to another object with some side s1 AGAINST side s2. Such a program is simpler and does not require great expertise in computer programming. In general, the task-level programs that have been developed so far have been applied to robot models. Some examples of such programs are given in [22 - 24].

1.1.3 Potential Field Methods

These are numerical modelling methods in which potentials are built around obstacles. For example, the gripper and an obstacle may be modelled as being positively charged. The goal to which the robot is moving is modelled as being negatively charged. As the obstacle repels the gripper, the

goal attracts it. The repulsion experienced by the gripper is large close to the obstacle. It dies off rapidly as the separation between the gripper and the obstacle increases. The attraction or repulsion has been expressed in terms of different parameters. For instance, Khatib's potentials were in terms of real space while Warren's were in c-space [14, 25 - 29].

Potentials have been used with some success. However, one faces local minima problems in using them. This means the gripper may fail to reach the goal when the work space is cluttered with obstacles because it is trapped in a potential trough between two or more obstacles. Furthermore, for obstacle avoidance involving all the links of a manipulator the model runs into the snag of repulsion between the links because of the same kind of charge placed on them as well as the obstacle. Yet physically the links have to remain attached at their joints. Also, while the use of circles or spheres to model the gripper may be reasonable, it is not realistic to use them for other links which are much longer [6, p 409]. An improved potential field method has to be used in such cases.

Other Penalty Functions

Apart from the use of potentials, penalty functions have been defined in other ways. For example, a safety function has been proposed for a mobile robot [30]. The workers quantified safety in terms of local information. The path they sought was the one that maximised the product of safety and attraction of the robot towards the goal.

1.1.4 Work Space Mapping

Many methods have been proposed to map out the obstacle-free subset of the work space of robotic manipulators. Some of these methods are now described.

Free Space Decomposition

In this method, the free space is divided up into a finite number of subsets which guarantee that the robot's configuration constraints are not violated. The subsets are then searched in turn to achieve path optimisation. Different shapes have been tried for these subsets. Examples of such shapes include cylinders and cones [31, 32].

Quadtrees, Octrees and Topology

In this case, the robot's work space is divided up into squares for two-dimensional problems and into cubes for three-dimensional problems. Each square or cube is further sub-divided repetitively into four squares or eight cubes until the homogenous squares or cubes in the free space can be determined. The homogeneous squares or cubes are those that do not overlap with any obstacle in the work cell. Topology is then employed to find a path between a start point and the goal within the homogeneous squares or cubes that have been found. As an example, Mehrotra and Krause [33] used a quadtree coding method for planning safe paths for a mobile robot. The smallest quad size used was smaller than the robot size. Thus the maximum depth of search in a tree was limited to the level at which the quad size was the same as that of the robot. Other workers have reported on their work based on this method [34 - 36].

Other Mapping Methods

Some other methods have been proposed. One is a method based on the relative displacements of the objects in the work volume. Choi et al. have proposed a mapping which transforms the relative displacements of an object in the work cell of a robot with respect to a reference object. They suggested that the generated image points could be used in detecting interference between the objects and in obstacle avoidance [37, 38].

Another method is the Voronoi Diagram [39]. The diagram on which the robot path is based is the locus of points which are equidistant from two or more obstacles in the immediate vicinity of a point of interest (the gripper origin say). The diagram's number of dimensions is one less than the number of dimensions of the robot work space. Thus a 3-D work space is mapped to a 2-D diagram.

Overall, potential methods have been found to be generally faster than mapping methods. On the other hand, mapping methods are detailed and tend to guarantee that a path can be found much more than potential methods can. Mapping methods do not suffer from the local minima problem faced when using potentials. As they are based on elemental portions of the work space, however, they are computationally expensive.

1.1.5 Sensing Devices

It is important to communicate information about the location of an obstacle to a robot. This is done by employing sensors or by explicitly stating the location of the obstacle in terms of a set of axes and time. The sensors which have been used include: (a) infra red sensors, (b) ultra sonic sensors and (c) vision cameras [40 - 43].

These sensors are usually mounted on the gripper in the case of manipulators. They can therefore detect possible collisions of a small part of the robot (e.g the gripper) with an obstacle. - The use of sensors when collision is to be avoided for every link of a robot is clearly a difficult proposition. To avoid collisions for all the links of the robot, a very large number of sensors would be required. Furthermore, sensors are expensive and fragile. They cannot withstand for long the harsh industrial environment and the dynamic forces suffered by robot links. If they fail in operation, damage may be caused to the robot and the sensor as

well. In contrast, there are no sensors to be damaged when using either potential or mapping methods for path planning. However, sensors are required for a robot work cell in which the obstacles are not static. Potential or mapping methods will fail in this case.

1.1.6 Path Optimisation

Path optimisation is an important aspect of robotic path planning. A robot must not just carry out its assigned task, it must do so efficiently. Using one of the methods outlined in the preceeding paragraphs, an initial path may be found. This path, which is called the first level path in this thesis, usually has many intermediate points between the start point and goal in comparison with what the path would have been in an obstacle-free cell. Using some criterion, the path has to be optimised to obtain a second level path which has fewer intermediate points. In order to optimise the first level path, the following criteria have been applied: (a) minimum time, (b) minimum distance, (c) minimum energy and (d) combination methods.

For the minimum time criterion, it is desired that the robot should get to the goal in the shortest possible time. Several algorithms have been based on this approach [37, 44 - 46]. The minimum time is subject to the maximum allowable torque or force at the robot joints. This minimum time should not be too small since excessive joint torques or forces will be developed. Neither can too great a time period justify the use of a robot which is meant to ensure increased productivity. Minimum time algorithms are based on some intermediate position between these two extremes.

The minimum distance method has also been used by some workers [47, 48]. Usually, in 2-D, the method entails enclosing the obstacles within rectangular regions and moving

the payload in a straight line to the closest rectangle corner. The corner must be one that makes getting the payload to the goal a possibility. If some other shape (for example a circle) is used to enclose an obstacle, it will be required that the path be tangential to the circle. Visibility graphs are employed in the minimum distance method [49]. For his work, Cao [47] applied Bellman's optimality principle in his own path planning algorithm applied to the PUMA 560 robot. He considered only payload collisions in a two-dimensional space.

While the minimum distance approach is basically a kinematic method, the minimum energy approach takes the dynamics of the robot system into consideration. This greatly complicates the optimisation algorithm. The approach has been applied to simpler robotic systems [50, 51].

Some other methods or a combination of two or more of the methods listed above have been used or proposed by some workers. Examples are methods based on design optimisation, time-energy, expert systems, and speed [52 - 56]. Also in their model, Dupont and Derby proposed a heuristic/free space method to find workable paths (not necessarily optimal) [57]. Further, variational principles have also been employed [58]. Minimisation of a cost function has been applied to some industrial SCARA robots [59]. A method for path planning based on the optimisation of an integral cost function has also been proposed [60]. The magnitude of this function is accumulated from the start of a move to some current position. This integration method contrasts with the more common approach of path planning based only on the information available at the instant that a collision is detected.

It should be noted that using different criteria will probably not yield exactly the same "optimal" paths for a given robot task. This arises from the fact that there is an inter-play of very many conflicting factors. Researchers have also defined their functions in the way that best fits their own investigations.

1.2 Thesis Outline

From the references given in Section 1.1 it would be seen, in general, that attempts to solve the path planning problem in the presence of obstacles have been done for simpler cases. Examples of such cases are:

(i) consideration of planar revolute robots with two or three joints,

(ii) solution of two-dimensional problems or, if three-dimensional, restriction of problems to prismatic manipulators,

(iii) path planning only for the gripper and payload combination with other links of the robot being neglected,

(iv) neglecting the possibility of relative rotation between a robot link and an obstacle when the world model entails shrinking the former and growing the latter.

The work reported in this thesis is an extension of solutions to the path planning problem and a step towards increased safety in robot work cells. The Mitsubishi RM 101 robot was modelled. A description of this robot will be given in Section 2.1. Highlights of this research are as follows:

(i) consideration of all the five degrees of freedom of the revolute robot; rotation at any joint is not suppressed,

(ii) consideration of the whole three-dimensional work space of the robot,

(iii) path planning for the gripper, the payload and all the links of the robot,

(iv) consideration of fixed and unconnected obstacles which moderately clutter the work space,

(v) allowing for the possibility of relative rotation between links and the obstacles,

(vi) modelling a link as a line and an obstacle as a grown cuboid and employing world space geometry,

(vii) heuristic path planning by prescribing robot motion around a detected obstacle.

An obstacle avoidance program (OBSTAP) is described in this thesis. It examines the different possible collisions that can occur based on actual geometry and plans a path for such situations. The free space between the start and goal of a move is not mapped out in any way. OBSTAP falls into the category of off-line programs written in a robot-level language. QUICK BASIC was used to code the program.

In the current work, the location of an obstacle is not communicated to the robot using sensors. As the program is geometry-based, it is inexpensive in comparison with sensor-based systems. Obstacles are fixed and information on their location is supplied by the program user.

The scope of OBSTAP and the obstacle detection method are detailed in Chapters 2 and 3 respectively. Path planning is described in Chapter 4 and a discussion on the data compiled in actual tests of OBSTAP is given in Chapter 5. The conclusion from this work is drawn in Chapter 6 while the Appendix contains a User's Manual for OBSTAP.

CHAPTER 2

SCOPE OF OBSTAP

2.1 The Mitsubishi RM 101 Robot

This research was carried out on the Mitsubishi RM 101 robot. It is a training robot of the revolute type and it has five degrees of freedom (yaw articulation excluded). The reference configuration from which the joint angles are measured (i.e zero position) and the joint axes are shown in Fig. 2.1. The world axes X_0, Y_0 and Z_0 are located at the base of the robot body. Based on the orientations of the joint axes, the Denavit - Hartenburg parameters for the RM 101 are as presented in Table 2.1. How these parameters can be determined is detailed in the second chapter of [61].

The dimensions of the robot are as shown in Fig. 2.2. The locations of the stepping motors are also shown. The operation of these motors is based on joint-interpolated control [62, p10]. This means that the proportion between the rotations at any two joints remains constant from one step to another during a move. All the joints start to move and stop moving at the same time. Wrist pitch and roll are accomplished by a combination of the rotations of motors 4 and 5. Other joint rotations and the opening or closing of the gripper are controlled by independent motors.
The limits and angular turn per step for each joint are given in Table 2.2. Detailed specifications can be found in [62].

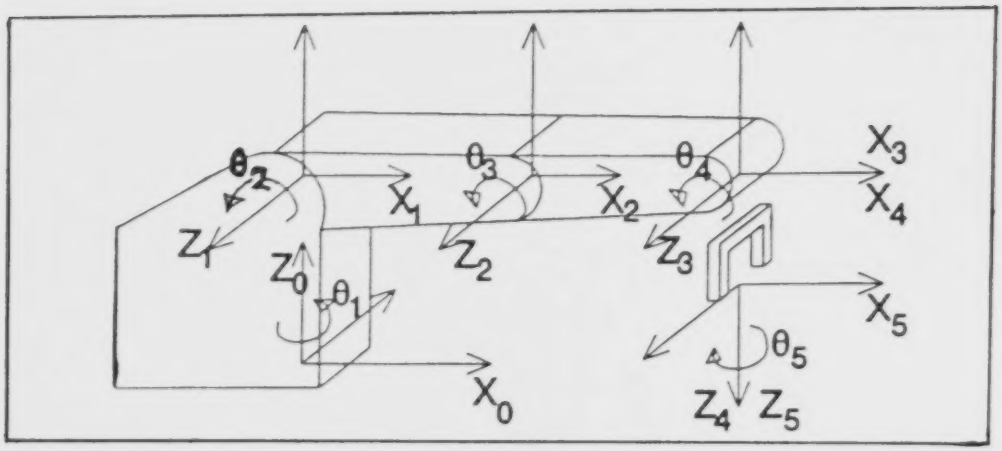


Fig. 2.1: Coordinate Frames For the RM 101 Robot.

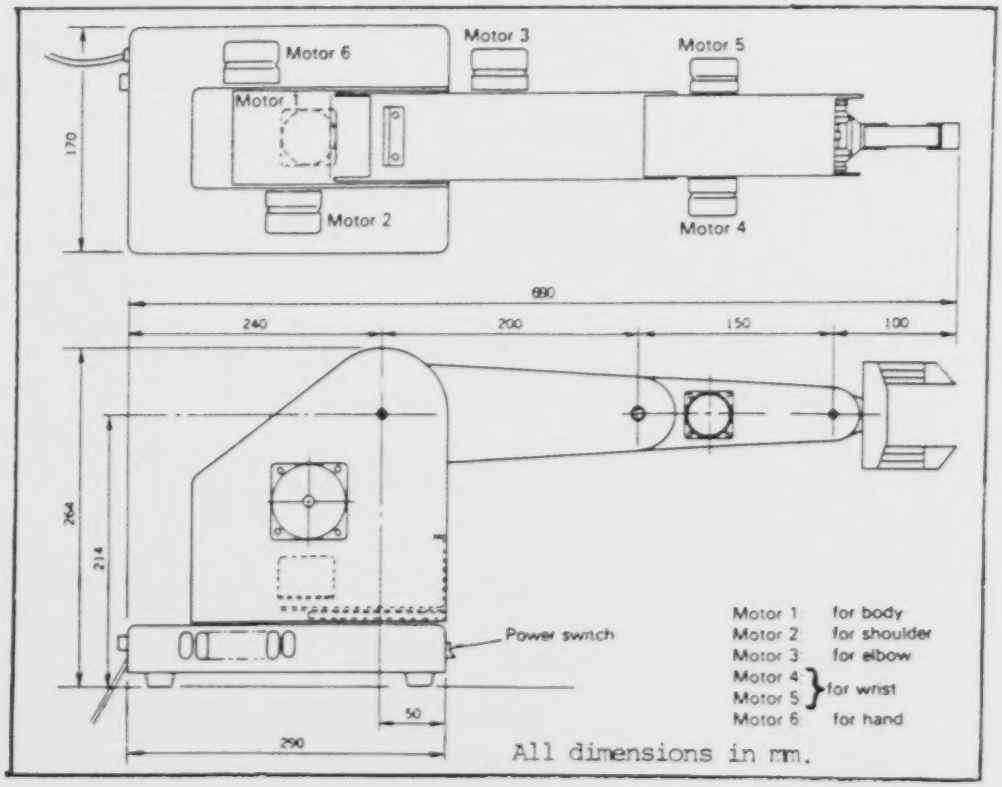


Fig. 2.2: Outlines and Dimensions of the RM 101 Robot.

Table 2.1: Denavit-Hartenburg Parameters For The RM 101.

Link	Joint Angle	Length (mm)	Offset (mm)	Twist (deg)
1. Body	θ_1	0	214	90
2. U.A	θ_2	200	0	0
3. F.A	θ_3	150	0	0
4. Pitch	θ_4	0	0	90
5. Roll	θ_5	0	85	0

U.A = Upper arm, F.A = Forearm

Table 2.2: RM 101 Joint Angle Limits and Step Sizes.

Joint	Min Angle (deg)	Max Angle (deg)	Step Size (deg)
Body (M1)	- 120	+ 120	0.04
Shoulder (M2)	- 30	+ 120	0.04
Elbow (M3)	- 120	0	0.08
Pitch (*)	0	+ 180	0.05
Roll (*)	- 180	+ 180	0.05
Gripper (M6)	- 400 (open)	0 (closed)	0.2

Angles are measured from the position shown in Fig. 2.1.
M1 = Motor #1 shown in Fig. 2.2. This motor alone causes the body to turn. Similarly for other motors.
(*) means pitch and roll are achieved by a combination of motors M4 and M5.

2.2 Uses of OBSTAP

The program OBSTAP has about 8,100 lines of code. It requires some 283 kbytes of memory and is run in the QUICK BASIC environment. OBSTAP can be used for several purposes which are briefly described in this section. A more detailed User's Manual is contained in the Appendix.

The program can be run to enter data which describes the user-specified moves to be made by the robot. These are called primary moves in the rest of this thesis to distinguish them from secondary moves which the program itself generates. A number of sequential primary moves constitute a task. OBSTAP can also be run to enter data which describe the obstacles in the work cell. Any data entered for the first time is saved so that it need not be entered from the keyboard for subsequent runs. In addition, OBSTAP can be run in order to carry out obstacle detection with or without path planning around the detected obstacle. The user also has the option of compiling the obstacle data being entered for the first time for animation purposes.

Robot Task Data

A primary move is specified in terms of the start and end points of the move. Successful path planning around an obstacle always gets the robot to the end point (or goal) from the start point. It is not allowed that either of these points should be lost.

A task of up to twenty primary moves can be handled by the program. However, when it is to be run on small RAM machines, fewer primary moves should be considered. This will help to prevent memory problems. A mere roll or the opening/closing of the end effector does not constitute a

primary move requiring obstacle detection and path planning.

The program user can use one of three methods to compile the task file data. They are:

(i) entering the gripper coordinates and orientations at the goals of the primary moves,

(ii) entering the joint angles values at the goals,

(iii) including the moves in a motor file [62].

This means that the description of a primary move can be in terms of either forward kinematics or inverse kinematics. A motor file consists of control commands specified by the manufacturers of the RM 101.

Obstacle Data

The data which describes an obstacle can be stored in a file by running OBSTAP. This is called the obstacle file in the rest of this thesis. The obstacle is represented in a simplified form as a cuboid. There are three dimensions (length, width and height) to work with. In addition, the sides of the cuboid may be parallel to the world axes or inclined at some acute angle to them. This means OBSTAP has much more flexibility in the process of growing obstacles than is found in using circles or spheres to enclose obstacles. This flexibility makes for a more efficient use of the robot work space. The program has been written to handle cases of moderate cluttering in which up to four obstacles are scattered within the work space. However, more than four obstacles were used in testing OBSTAP.

Obstacle Detection Only

The program user may investigate whether or not a particular primary move will cause a collision. In this case, OBSTAP does not plan a path around any detected obstacle since only information on collision is what is wanted. Options are available for either checking for just one particular move of interest among a set of primary moves in the task file or

checking for all the moves in the file. When there is a collision, the information supplied by OBSTAP identifies the particular primary move, the obstacle, the link involved [upper arm, forearm or gripper] and the particular side of the obstacle [top, bottom or a vertical side]. In addition, the joint angle values, the gripper position and the collision point coordinates are supplied.

Obstacle Detection With Path Planning

Obstacle detection followed by path planning around a detected obstacle can also be done using OBSTAP. The program user may choose to observe the planned first level path as it is determined step by step. In this case the robot has to be connected to the computer. Alternatively he may compile motor files for the first level and second level paths. These can be downloaded to the robot at a later stage. OBSTAP prompts the user when a path has been found for a primary move and also returns the time taken to complete computations for all the primary moves.

Geometry Simplification

In order to have a simplified geometric model, each link of the robot is shrunk to a line when carrying out either obstacle detection alone or path planning around a detected obstacle. Correspondingly, the obstacles in the work cell have to be grown. The geometric relations of lines and planes are then used in the program to check for collisions. The default dimension by which obstacles are grown is 50 mm. This value is based on the largest gripper opening. However, at a prompt, the user may vary this dimension if he so wishes.

Animation

When the data describing an obstacle is being entered for the first time, the program user may specify that animation data should be compiled. No animation routine is

contained in OBSTAP itself. However, it can compile work cell data that can be used in the program ANIM written by Toogood [63]. The advantage of this to OBSTAP is that prior to downloading a compiled motor file to the robot, the planned path can be viewed on a monitor. This will give an indication of how safe the path is.

CHAPTER 3

OBSTACLE DETECTION

In this chapter, the mathematical expressions on which obstacle detection in OBSTAP is based are presented. Symbols which are printed bold and large represent vectors or matrices.

3.1 Geometric Tools

Geometric relationships that exist in cartesian world space between points, lines and planes were used in coding OBSTAP. The homogeneous matrix forms of these relationships are detailed in the first chapter of [64]. The relationships are briefly presented in this section.

A point $\mathbf{U}$ which has coordinates x, y and z is represented in the form:

$$\boldsymbol{U} = [x \;\; y \;\; z \;\; 1]^T \qquad (3.1)$$

The superscript T indicates a transpose. The value 1 is included by convention in the 4 x 4 matrices which are usually encountered in the analysis of robot systems.

A line $\mathbf{L}$ can be defined in terms of two points $\mathbf{U}_1 = [x_1 \;\; y_1 \;\; z_1 \;\; 1]^T$ and $\mathbf{U}_2 = [x_2 \;\; y_2 \;\; z_2 \;\; 1]^T$ on the line. Let $\mathbf{W}(\alpha)$ represent a general point on the line. We have:

$$\boldsymbol{W}(\alpha) = \boldsymbol{U}_1 + \alpha(\boldsymbol{U}_2 - \boldsymbol{U}_1) \qquad (3.2)$$

where α is some parameter satisfying $-\infty \leq \alpha \leq \infty$. At the point $\mathbf{U}_1$ the value of α is 0 and at $\mathbf{U}_2$ the value is 1.

A plane $\mathbf{P}$ may be defined as a row vector in terms of

its unit normal vector $a\underline{i} + b\underline{j} + c\underline{k}$ which is localised at the point of minimum distance d of the plane from the origin. We have:

$$\boldsymbol{P} = [a \;\; b \;\; c \;\; d] \tag{3.3}$$

Consider the minimum distance between two lines $\mathbf{L}_1$ and $\mathbf{L}_2$. Let $\mathbf{L}_1$ be defined in terms of points $\mathbf{U}_1$ and $\mathbf{U}_2$ and a parameter α. $\mathbf{L}_2$ is defined in terms of points $\mathbf{U}_3$ and $\mathbf{U}_4$ and a parameter β. Let the distance from a point on the first line to a point on the second be denoted by D_1. When this distance is minimum we have:

$$\frac{\partial D_1}{\partial \alpha} = 0 \tag{3.4}$$

$$\frac{\partial D_1}{\partial \beta} = 0 \tag{3.5}$$

The minimum distance of the point $\mathbf{U}$ (Eqn. 3.1) and the plane $\mathbf{P}$ (Eqn. 3.3) is the distance between $\mathbf{U}$ and the point of intersection of the plane and the normal from $\mathbf{U}$. Denoting this intersection point as $\mathbf{V}$ we have:

$$\boldsymbol{V} = \boldsymbol{U} - \boldsymbol{PU}[a \;\; b \;\; c \;\; 0]^T \tag{3.6}$$

Let the distance between $\mathbf{U}$ and $\mathbf{V}$ be D_2 in this case.

The point of intersection of the line $\mathbf{L}$ (Eqn. 3.2) and plane $\mathbf{P}$ (Eqn. 3.3) is the point on the line for which we have:

$$\alpha = -\frac{\boldsymbol{PU}}{\boldsymbol{P}(\boldsymbol{U}_2 - \boldsymbol{U}_1)} \tag{3.7}$$

Let D_3 represent the distance between the line and the plane.

If α is indeterminate the line and the plane are parallel. For this situation we have:

$$D_3 > 0 \qquad (3.8)$$

When the line and the plane are not parallel we have:

$$D_3 = 0 \qquad (3.9)$$

3.2 Growth of Obstacles

Consider the bar of length L shown in Fig. 3.1. By being stretched equally at both ends its centroid G is not displaced. Let each end be displaced by $\delta L/2$. The new position x′ of an element initially located at a distance x from the centroid is given by:

$$x' = x(1 + \frac{\delta L}{L})$$

This idea was applied to the growth of obstacles in OBSTAP. Fig. 3.2 shows an obstacle initially having dimensions 2a, 2b and 2c respectively in the X, Y and Z directions of its centroidal cartesian system. Axis Z is parallel to the world Z_0 axis but respectively axes X and Y need not be parallel to X_0 and Y_0. Thus the vertical sides of the obstacle may be inclined at acute angles to X_0 and Y_0. The position of the centroid with respect to the world origin is given by coordinates x_G, y_G and z_G. The obstacle is grown round about by a growth dimension g_d. A vertex of the obstacle initially having coordinates x, y and z with respect to the centroid becomes displaced to a new position having coordinates x′, y′and z′ also with respect to the centroid.

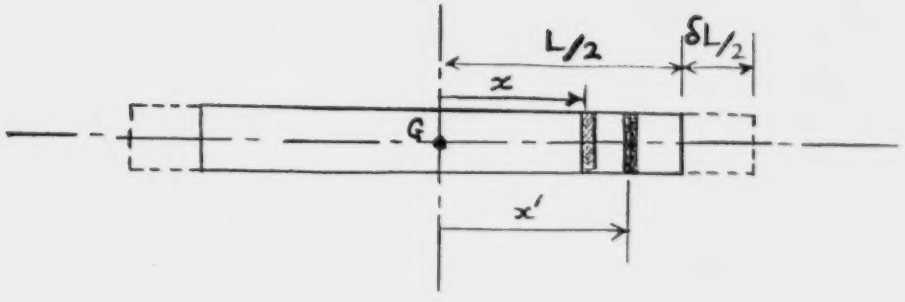
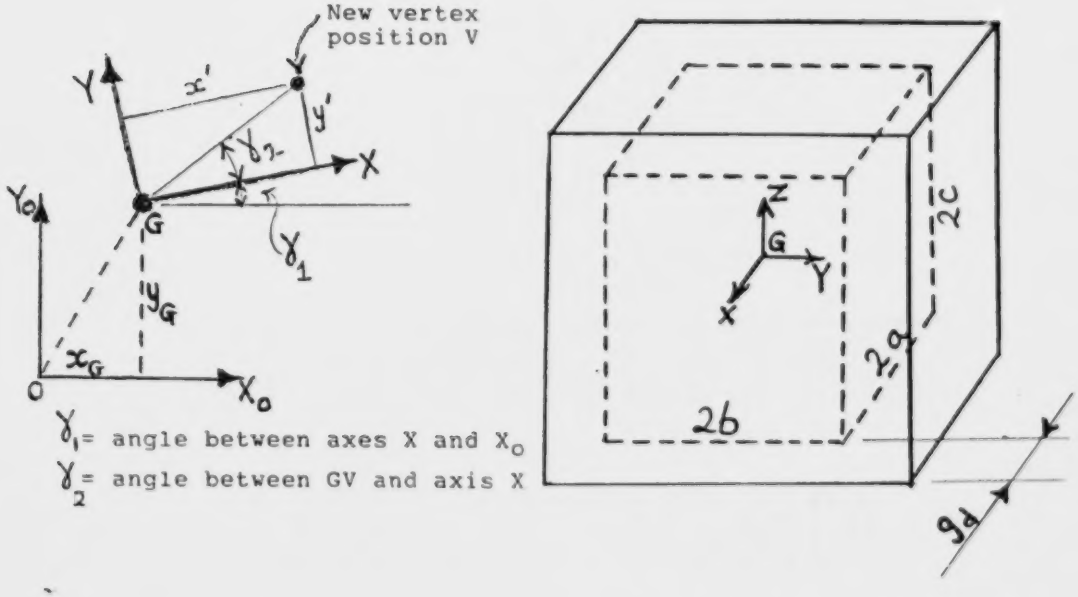


Fig. 3.1: Stretching a Bar Equally at Both Ends.



(a) Centroidal Axes Relative to World Axes.

(b) Obstacle Growth Around the Centroid.

Fig. 3.2: Growing an Obstacle Represented as a Cuboid.

We have:

$$x' = x + \frac{(x - x_G)}{|x - x_G|} g_d \tag{3.10}$$

$$y' = y + \frac{y - y_G}{|y - y_G|} g_d \tag{3.11}$$

$$z' = z + \frac{z - z_G}{|z - z_G|} g_d \tag{3.12}$$

The coordinates of the new position of the vertex with respect to the world origin, x'', y'' and z'', are then given by:

$$[x'' \;\; y'' \;\; z'' \;\; 1]^T = [x_G + A \;\; y_G + B \;\; z_G + z' \;\; 1]^T \tag{3.13a}$$

With reference to Fig. 3.2, the values of A and B are as follows:

$$A = |(x'^2 + y'^2)^{\frac{1}{2}}| \cos(\gamma_1 + \gamma_2) \tag{3.13b}$$

$$B = |(x'^2 + y'^2)^{\frac{1}{2}}| \sin(\gamma_1 + \gamma_2) \tag{3.13c}$$

3.3 Instantaneous Robot Position

Considering a primary move, let θ_s represent the set of joint angles at the start of the move while θ_g represents the

set at the goal. A general joint angle is represented by θ. The body joint angle is identified by subscript 1, the shoulder joint angle by 2 and the elbow joint angle by 3. Pitch and roll angles have subscripts 4 and 5 respectively.

$$\boldsymbol{\theta}_s = [\theta_{s1} \ \theta_{s2} \ \theta_{s3} \ \theta_{s4} \ \theta_{s5}] \tag{3.14}$$

$$\boldsymbol{\theta}_g = [\theta_{g1} \ \theta_{g2} \ \theta_{g3} \ \theta_{g4} \ \theta_{g5}] \tag{3.15}$$

To implement joint-interpolated control, the joint angle which will change the most either positively or negatively is determined at the start of the primary move. The value of this angle is then varied in steps of $\pm$ 1 degree depending on whether the value at the goal is less or greater than the value at the start. The values of the remaining joint angles are found by proportion. Consider an example in which the joint i is to turn the most at the start of a primary move and n computation steps have been taken without a collision being detected. We have for the joint:

$$\theta_i = \theta_{si} \pm n \tag{3.16}$$

For any other joint j we have:

$$\theta_j = \theta_{sj} \pm n \frac{\theta_{gj} - \theta_{sj}}{\theta_{gi} - \theta_{si}} \tag{3.17}$$

From the joint angles, all the Denavit-Hartenburg parameters and the elements of the **A** matrix for each link can be determined [61, chapter 2]. The $\mathbf{T}_5$ matrix which transforms the world cartesian system to that at the gripper origin is given by:

$$\boldsymbol{T}_5 = \boldsymbol{A}_1 \boldsymbol{A}_2 \boldsymbol{A}_3 \boldsymbol{A}_4 \boldsymbol{A}_5 \qquad (3.18)$$

Here subcripts 1, 2 and 3 stand for the body, upper arm and forearm respectively. The links representing gripper pitch and roll are denoted by the subscripts 4 and 5 respectively.

Similarly the **T** matrices which transform the world axes into the axes at the different joints (Fig. 2.1) are as follows:

$$\boldsymbol{T}_4 = \boldsymbol{A}_1 \boldsymbol{A}_2 \boldsymbol{A}_3 \boldsymbol{A}_4 \qquad (3.19)$$

$$\boldsymbol{T}_3 = \boldsymbol{A}_1 \boldsymbol{A}_2 \boldsymbol{A}_3 \qquad (3.20)$$

$$\boldsymbol{T}_2 = \boldsymbol{A}_1 \boldsymbol{A}_2 \qquad (3.21)$$

$$\boldsymbol{T}_1 = \boldsymbol{A}_1 \qquad (3.22)$$

The elements $\mathbf{T}_5(1,4)$, $\mathbf{T}_5(2,4)$ and $\mathbf{T}_5(3,4)$ are respectively the instantaneous x, y and z coordinates of the gripper origin with respect to the world axes. The instantaneous x, y and z coordinates of the joints are similarly found. Thus each link can be modelled as a straight line in terms of the coordinates of its extreme points.

3.4 Collision Checks

From the obstacle data entered for a particular program run, OBSTAP computes and stores data describing all the six sides of a grown obstacle as planes. As described in the last section, the program models the links as straight

lines. OBSTAP then uses the geometric relationships given in Section 3.1 to check for collisions.

A criterion has been set to determine if a collision is imminent. The criterion is that the least distance between a link and an obstacle should be less than or equal to 16 mm. This value corresponds approximately to the distance by which the gripper origin moves when the body turns by 2 degrees with the arm fully extended. Collision checks are made for each link and for the faces and edges of each potential obstacle.

There are three main collision types. They are:

(i) collision between a link and an obstacle edge,

(ii) collision between a point (e.g. a joint) and a face,

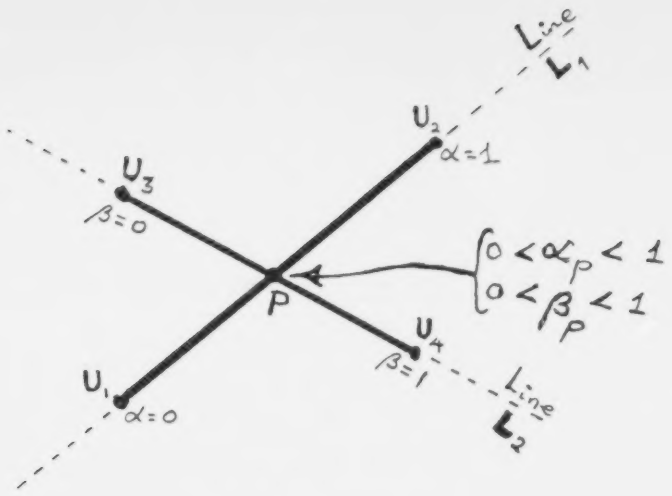
(iii) collision between a link and a face.

The first case is modelled as collision (or intersection) between two lines, the second as collision between a point and a plane and the third as collision between a line and a plane. Suppose there is a collision for a given robot configuration. Then for the obstacle face for which computations are being carried out we have the following necessary but not sufficient condition:

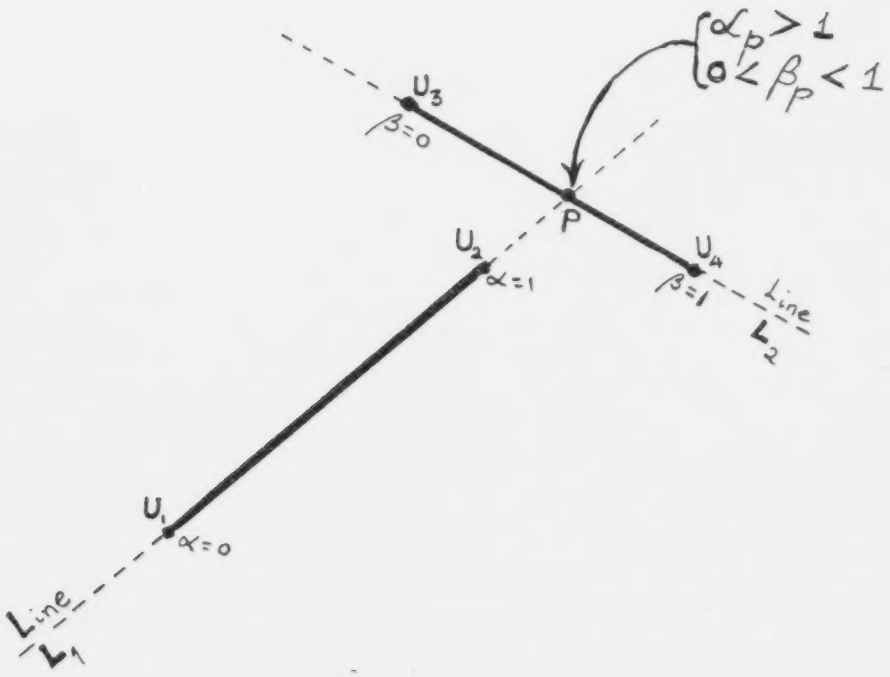
$$\min(D_1,\ D_2,\ D_3) \ \leq\ 16\ mm \qquad (3.23)$$

Other special collision cases may arise. Examples of these special cases include collision between a joint and an edge or collision between a link and a face parallel to the link. In coding OBSTAP it was assumed that the probability of the occurence of such special cases would be very low. It was also assumed that even if a special collision case arises when using the program one of the three main collision types listed above would be able to trap the collision. It was assumed that the types are general enough.

The links are not modelled as infinitely long lines and



(a) A Case of True Collision.



(b) A Case of Untrue Collision.

Fig. 3.3: Checking for Collision Between Two Line Models

the obstacle faces are not modelled as infinitely large planes. Because of this, checks are carried out within the collision routine to confirm whether or not a suspected collision point (which satisfies Eqn. 3.23) is truly a collision point.

Two lines are shown in Fig. 3.3 to illustrate the first type of collision. We assume that $\mathbf{L}_1$ has been defined in terms of points $\mathbf{U}_1$ and $\mathbf{U}_2$ and a parameter α. Similarly $\mathbf{L}_2$ has been defined in terms of points $\mathbf{U}_3$ and $\mathbf{U}_4$ and a parameter β. A suspected collision point between the lines is truly a collision point providing the values of α and β at the point satisfy the following inequalities:

$$0 \leq \alpha \leq 1 \tag{3.24}$$

$$0 \leq \beta \leq 1 \tag{3.25}$$

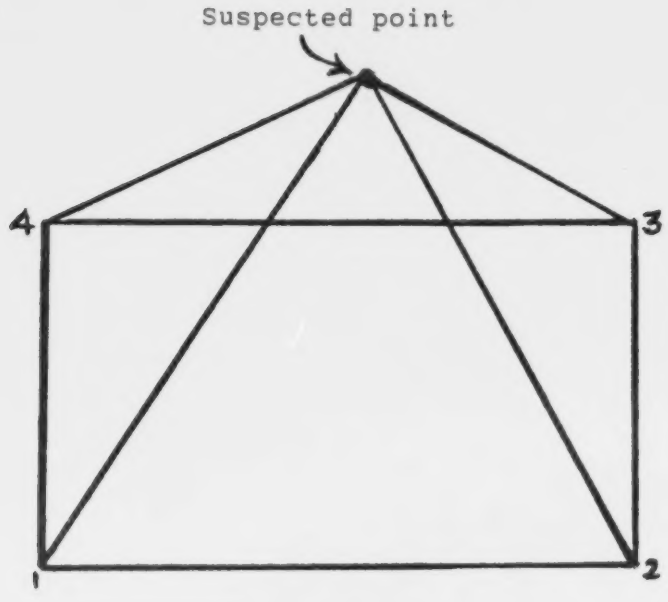
A suspected collision point, which the check proves not to be a true collision point, and a face of an obstacle are depicted in Fig. 3.4. The figure also shows a true collision point. This figure is applicable to the last two collision types. The areas a_1, a_2, a_3 and a_4 are formed in each case by joining the point with the four corners of the face using straight lines. If a represents the area of the face, the following conditions hold.

For an untrue point:

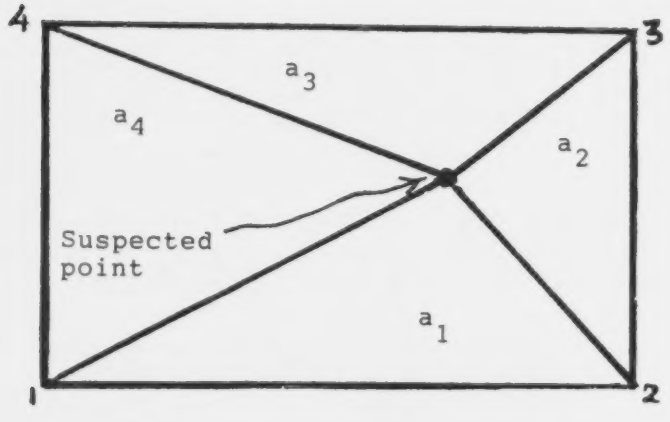
$$a_1 + a_2 + a_3 + a_4 > a \tag{3.26}$$

For a true point:

$$a_1 + a_2 + a_3 + a_4 = a \tag{3.27}$$



(a) No collision.



(b) A Case of True Collision.

Fig. 3.4: Checking for Collision Between (i) A Point and a Plane and (ii) A Line and a Plane.

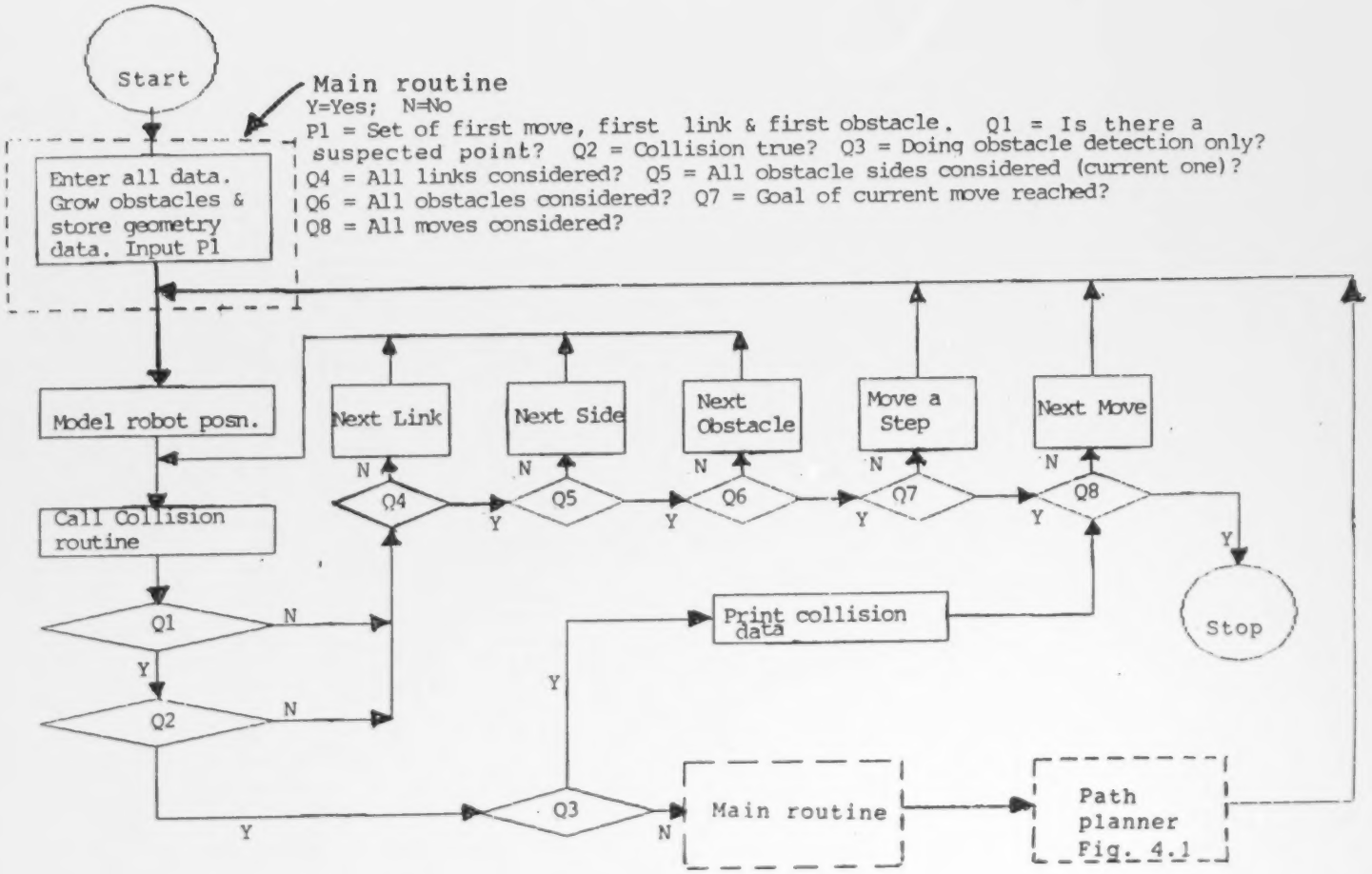


Fig. 3.5: Obstacle Detection Flow Chart

3.5 Detection Algorithm

Fig. 3.5 is a flow chart depicting the process of obstacle detection. After the robot task has been defined and the obstacle data entered, the following steps are taken in the process of obstacle detection:

(I) Grow the obstacles using the default growth dimension or the user-specified dimension (Eqn. 3.10 - 3.13).

(II) Determine geometry data for vertices, edges and faces of the grown obstacles (Eqn. 3.1 - 3.3).
Consider the robot's first primary move.

(III) (i) For the current primary move and robot position, determine the joint step sizes based on joint-interpolated control (Eqn. 3.14 - 3.17).
(ii) Model links as lines (Eqn. 3.18 - 3.22).

(IV) (i) Select a link and an obstacle.
(ii) Select one face of the obstacle.
(iii) Check for all three collision types between the link and the face (Eqn. 3.4 - 3.9, 3.23).

(V) IF there is a suspected collision from step (IV) (iii),
THEN check that the collision is true (Eqn. 3.24 - 3.27).

(VI) IF there is a suspected collision from step (IV) (iii) AND IF the collision is proved true in step (V),
THEN

(i) IF the program is being run for obstacle detection only,
THEN print collision data.

IF all the primary moves have not been considered,
THEN select the next primary move and GOTO step (III) (i);
ELSEIF all the primary moves have been considered,
THEN STOP.

(ii) IF the program is being run for the path planning option,
THEN RETURN to the main routine,
CALL path planner.
On RETURN GOTO step (III) (i).

(VII) IF there is no collision from step (IV) (iii) OR IF a suspected collision from step (IV) (iii) is proved false in step (V),
THEN

(i) IF all robot links have not been considered for the current face,
THEN select a new link and GOTO step (IV) (iii).

(ii) IF all robot links have been considered for the current face but all the obstacle faces have not been considered for the current robot position,
THEN select a new face and GOTO step (IV) (iii).

(iii) IF all the obstacles have not been considered for the current robot position,
THEN GOTO step (IV) (i).

(iv) Determine if the goal of the current primary move has been reached (the current robot

position and that given by Eqn. 3.15 will be the same).
IF the goal has not been reached,
THEN move the robot one step and GOTO step (III) (ii).

(v) IF all the primary moves have not been considered,
THEN consider next primary move and GOTO step (III) (i).
ELSEIF all the primary moves have been considered,
THEN STOP.

When it has been confirmed that a point is truly a collision point, the collision data is printed for the current primary move, if the user initially chose the option for obstacle detection without path planning. Otherwise path planning is commenced. This is presented in the next chapter.

CHAPTER 4

PATH PLANNING

4.1 Heuristic Search

Path planning in OBSTAP is based on heuristic search in world space around the detected obstacle. This consists of trying out a number of possible paths, detailed in Section 4.2, around the detected obstacle. Computations are carried out for the configuration of the robot relative to the obstacle along the trial path. A trial path is abandoned once it is found that a collision would occur along it. A new path is then sought.

The search is in world space. This is because the positions to which it is desired to move the robot can be easily imagined and specified in world space. It would be more difficult to imagine and specify these positions in some other spatial system (such as configuration space or some transform of the world space).

The path planning routine prescribes ways of moving the robot past the obstacle. The prescribed trial paths depend on the position of the robot relative to the obstacle at the time of collision. They also depend on the initial robot task - whether or not the body is moving.

Since a collision can occur in an infinite number of ways, heuristic search in robotic path planning may initially be considered as being unwieldy. In OBSTAP, however, similar collisions are grouped together and the rules that are prescribed are for these groups rather than for individual collisions. This approach has helped in achieving a manageable number of IF ... THEN statements in the path planning routine.

There may be several possible paths around an obstacle. OBSTAP does not explicitly attempt to find the optimal path

out of the set based, for example, on the minimum distance approach. The program simply seeks a path around the obstacle. Nevertheless, some optimisation is done in terms of computation time. As soon as a collision with an obstacle is detected, collision checks with other potential obstacles are abandoned. Path planning around the obstacle in the greatest danger of collision is commenced. The CPU time required to find the path is usually much smaller than that for exhaustive optimum path searches. This approach was taken in coding OBSTAP because it was felt that path planning around the detected obstacle should be regarded as being more paramount than completing collision computations. We may consider two obstacles A and B with which collisions may occur during a primary move. Let us assume that collision with A is more imminent. It is quite possible that obstacle B which initially posed a threat no longer does so after planning a path around A. However, this possibility is not guaranteed.

The methods of searching a few trial paths and of prescribing collision-dependent rules to move the robot have parallels in other areas of Engineering. An example of this is the use of rules of thumb in design. Sandgren and Venkataraman [52] have also proposed a solution to the path planning problem in 2-D space by considering only a discrete set of robot arm and gripper positions. This is reasonable in the light of the many conflicting factors associated. It can thus be seen that the approach taken in coding OBSTAP is not unusual.

4.2 First Level Path

There are three main ways by which a revolute manipulator that is similar to the RM 101 can move past an obstacle. Because the body can only rotate about the world Z axis, the three ways are:

(i) for some portion of the other links to be raised above the obstacle,

(ii) for some portion of the other links to be lowered below the obstacle,

(iii) for the other links to be retracted towards the body.

The first level path is based on the points listed above. It is impossible to have the body translate past the obstacle since we are not dealing with a mobile robot in this case. Fig. 4.1 is a flow chart showing the broad sub-divisions of the first level path planner. Details of these sub-divisions are given in the sections which follow.

The symbol $\mathbf{I}(\mathbf{U})$ stands for the inverse kinematic solution. It is used to determine the joint angles corresponding to a new gripper position $\mathbf{U}$ which is being tried. The gripper orientation at the current position is maintained. Thus for the new position being tried, the elements of $\mathbf{U}$ are entered into the $\mathbf{T}_5$ matrix such that with $1 \leq i \leq 4$ we have:

$$T_5(i,4) = U(i) \qquad (4.1)$$

We let the collision point be $\mathbf{P}$. The joint angles at the time of collision are respectively θ_{P1}, θ_{P2}, θ_{P3}, θ_{P4}, θ_{P5} for the body, shoulder, elbow, wrist pitch and wrist roll. Also at the time of collision, the gripper origin coordinates are x_{gr}, y_{gr} and z_{gr}. The coordinates at the desired goal for the primary move are x_g, y_g and z_g.

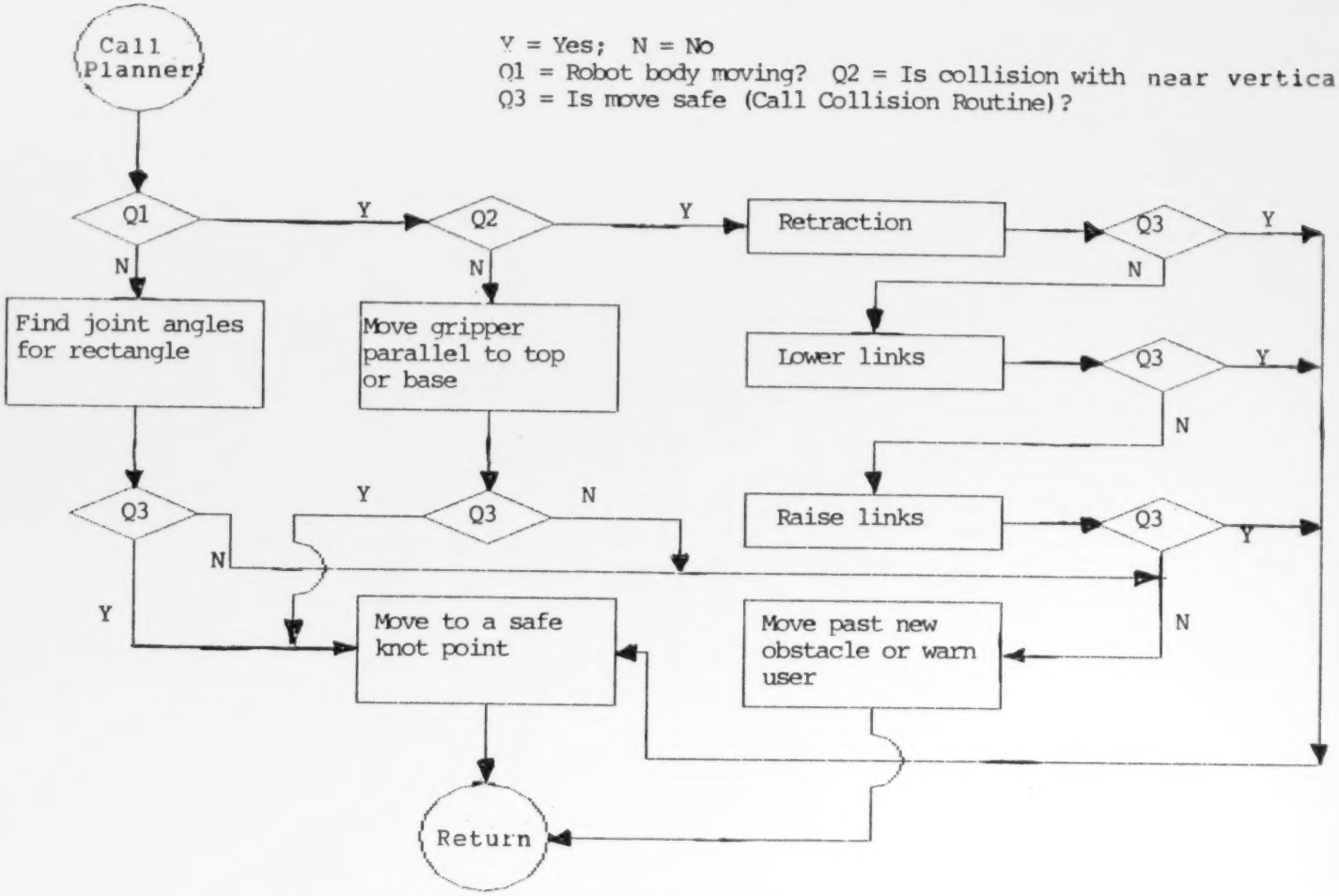


Fig. 4.1: Path Planning Flow Chart.

4.2.1 Cases of Variable Body Joint Angle

The upperarm, the forearm and the gripper all move in the same vertical plane. Without the turning of the body the gripper cannot reach every part of the work space. Thus, all the possible motions can be classified into two broad categories:

(i) rotation at the body joint with or without rotation at some other joint,

(ii) no rotation at the body joint with rotation at one or more of the other joints.

Path planning for the first category of robot task is dealt with in this sub-section.

4.2.1.1 Collision With a Vertical Near Side

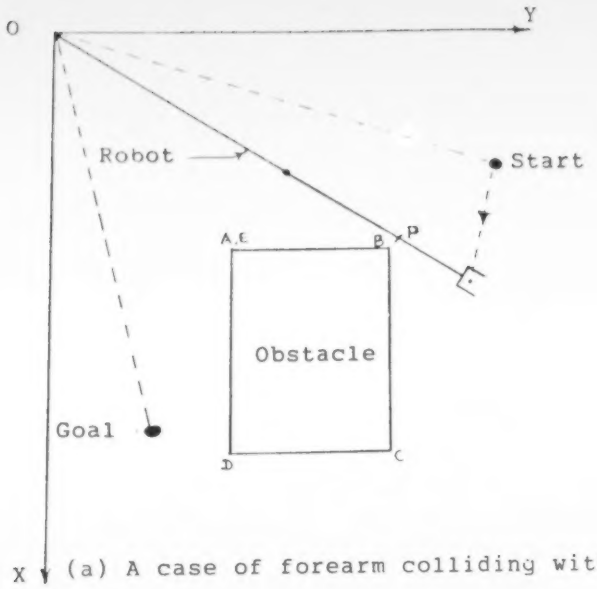
Fig. 4.2 depicts the case of a collision with a vertical side of an obstacle. The sides AB and AD which are near to the origin O are considered here. Only the plan view of the obstacle is shown in the figure.

The obstacle must be positioned somehow in the work space. Many times the table acts as the support, in which case the path to be planned cannot go below the obstacle. If the obstacle is suspended it may not be possible to plan a path which goes over the obstacle. In order to get the robot links past the obstacle, OBSTAP would do the following in turn until a path is found:

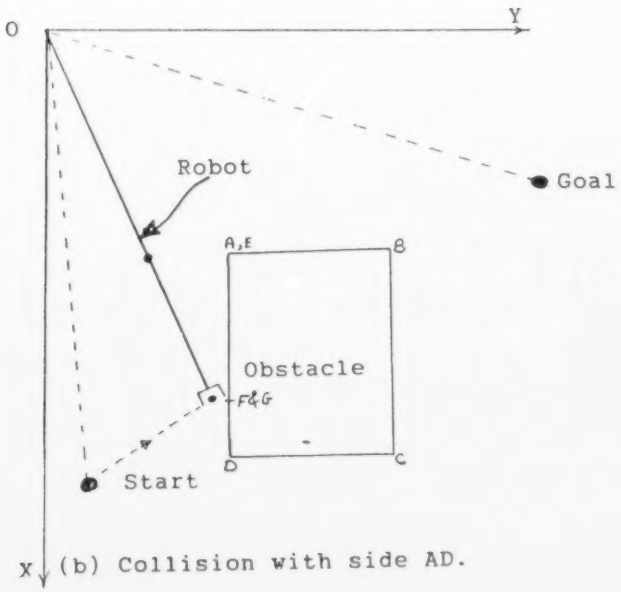
(i) Link Retraction

(a) Compute joint angles θ_{Ei} (where $1 \leq i \leq 5$) for some point **E** on the obstacle. Point **E** has coordinates x_E, y_E and z_E such that $x_E = x_A$, $y_E = y_A$ and $z_E = z_{gr}$. [This means OBSTAP carries out the **I**(**E**) computation].

(b) Call the collision routine to determine if there



(a) A case of forearm colliding with side AB.



(b) Collision with side AD.

Fig. 4.2: Collision With the Vertical Near Side of an Obstacle (Robot Body Moving).

would be a collision with another obstacle in moving the gripper from its current position to point **E**. The movement is in two stages of link retraction followed by body turn.

(c) IF SAFE, (1) retract links or write lines to PATH1.MOT such that the joint angles change by $(\theta_{Ei} - \theta_{Pi})$ where $2 \leq i \leq 5$.

(2) Turn robot body alone or write lines to PATH1.MOT such that the body joint angle changes by $\theta_{E1} - \theta_{P1}$.

(3) Move the robot to the desired goal using **E** as a new starting point.

(d) IF UNSAFE, try lowering links.

(ii) Lowering Links

(a) Carry out the **I**(**F**) computation. Point **F** has coordinates x_F, y_F and z_F such that $x_F = x_{gr}$, $y_F = y_{gr}$ and $z_F = z$ at obstacle base. Decrease z_F if necessary to ensure that the links are safely below the obstacle and carry out an updated **I**(**F**) computation.

(b) Call the collision routine to determine if there would be a collision with another obstacle or the work top in moving the gripper from its current position to point **F**.

(c) IF SAFE, (1) Lower the links or write lines to PATH1.MOT such that the joint angles change by $(\theta_{Fi} - \theta_{Pi})$ where $1 \leq i \leq 5$. (Robot body does not move).

(2) Move the robot to the desired goal using **F** as a new starting point.

(d) IF UNSAFE, try raising links.

(iii) Raising Links

(a) Carry out the **I**(**G**) computation. Point **G** has coordinates x_G, y_G and z_G such that $x_G = x_{gr}$, $y_G = y_{gr}$ and $z_G = z$ at obstacle top. Increase z_G if necessary to ensure that the links are safely above the obstacle and carry out an updated **I**(**G**) computation.

(b) Call the collision routine to determine if there would be a collision with another obstacle in moving the gripper from its current position to point **G**.

(c) IF SAFE, (1) Raise links or write lines to PATH1.MOT such that the joint angles change by $(\theta_{Gi} - \theta_{Pi})$ where $1 \leq i \leq 5$. (Robot body does not move).

(2) Move the robot to the desired goal using **G** as a new starting point.

(d) IF UNSAFE, the user is warned that the robot may be trapped. He is advised to try another growth dimension for the run.

OBSTAP first attempts to retract links. If it is found that this would fail, an attempt is made to lower or raise the links. This does not confer prime importance on the retraction process over and above lowering or raising the links. It is just that one set of computations have to be done first before the next.

Lowering the links is somewhat tricky. This is because by merely positioning the gripper below an obstacle, it is not guaranteed that the upperarm and the forearm are safely under the obstacle. An obstacle has to be placed very high in the work space for this condition to exist. Another factor that complicates path planning by lowering the links is that the table itself becomes a potential obstacle. However, because the computation for raising the links follows that of lowering

them, there is still a way out and the robot can be saved a possible collision with the table.

4.2.1.2 Collision With Top or Bottom or Vertical Far Side

Another possibility is that of a collision with the top or bottom or a vertical side which is far from the origin. A plan view of such a configuration is shown in Fig. 4.3. In this case, an attempt is made to move the gripper parallel to the obtacle top or bottom. The primary move starting from S1 should be considered for the collision involving either the top or bottom. The primary move starting from S2 should be considered for the collision involving either side BCJI or side CDKJ. OBSTAP would do only one of the following:

(i) Top Side (ABCD)

(a) Carry out the **I**(**G**) computation as before.

(b) Determine the body joint angle for the obstacle top corner point **D**.

(c) Check if there would be a collision with another obstacle when moving the robot at the body joint alone until the vertical plane through **D** is reached.

(d) IF SAFE, (1) Move links or write lines to PATH1.MOT such that joint angles change by $(\theta_{Gi} - \theta_{Pi})$ where $2 \leq i \leq 5$. (Body fixed).

(2) Move links or write lines to PATH1.MOT such that only the body joint angle changes by $(\theta_{D1} - \theta_{P1})$.

(3) Move the robot to the desired goal using **D** as a new starting point.

(e) IF UNSAFE, try moving past new obstacle (most probably

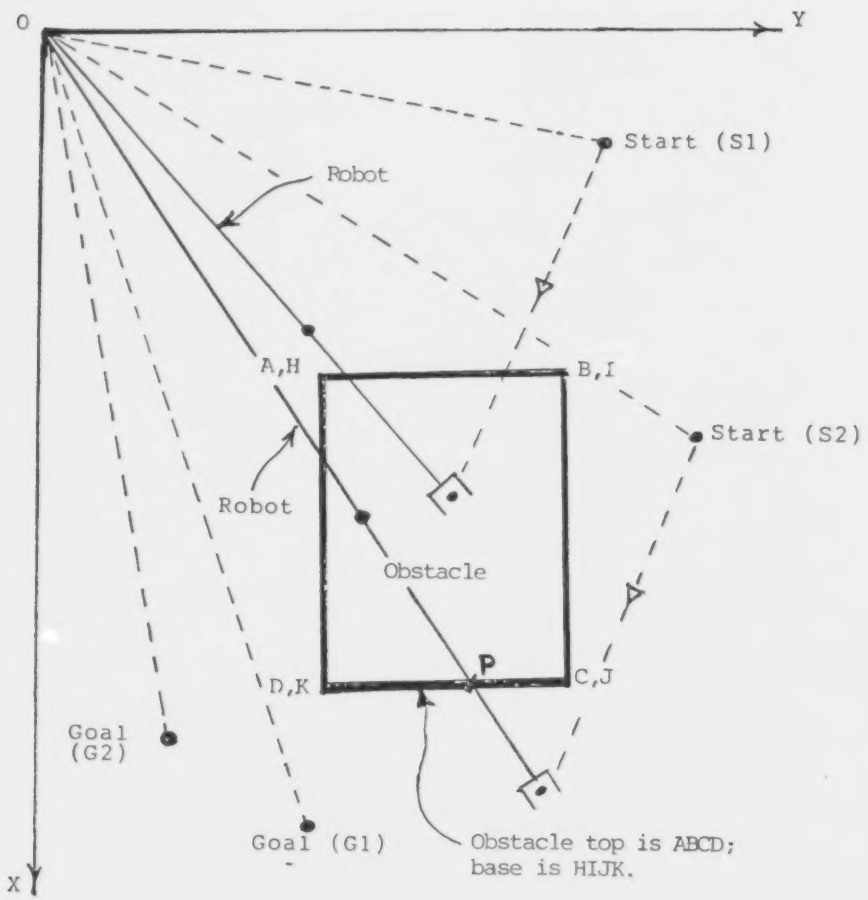


Fig. 4.3: Collision With the Top or Bottom or a Vertical Far Side of an Obstacle (Robot Body Moving).

the collision would have occurred with one of its vertical sides).

(ii) Bottom Side (HIJK)

(a) Carry out the **I**(**F**) computation as before.

(b) Find the body joint angle for the base corner point **K**.

(c) Check if there would be a collision with another obstacle when moving the robot at the body joint alone until the vertical plane through **K** is reached.

(d) IF SAFE, (1) Move links or write lines to PATH1.MOT such that joint angles change by $(\theta_{Fi} - \theta_{Pi})$ where $2 \leq i \leq 5$. (Body fixed).

(2) Move links or write lines to PATH1.MOT such that only the body joint angle changes by $(\theta_{K1} - \theta_{P1})$

(3) Move the robot to the desired goal using **K** as a new starting point.

(e) IF UNSAFE, try moving past new obstacle (most probably the collision would have occurred with one of its vertical sides).

(iii) Vertical Far Side (BCJI or CDKJ)

IF a part of the robot is directly above the obstacle, THEN treat the case as for top side ABCD above.

IF a part of the robot is directly below the obstacle, THEN treat the case as for bottom side HIJK above.

4.2.2 Cases of Fixed Body Joint Angle

Typical collisions for this category of robot task are represented in Figures 4.4 and 4.5. Rectangles LMNO and QRST are extended sections through the obstacle with which the

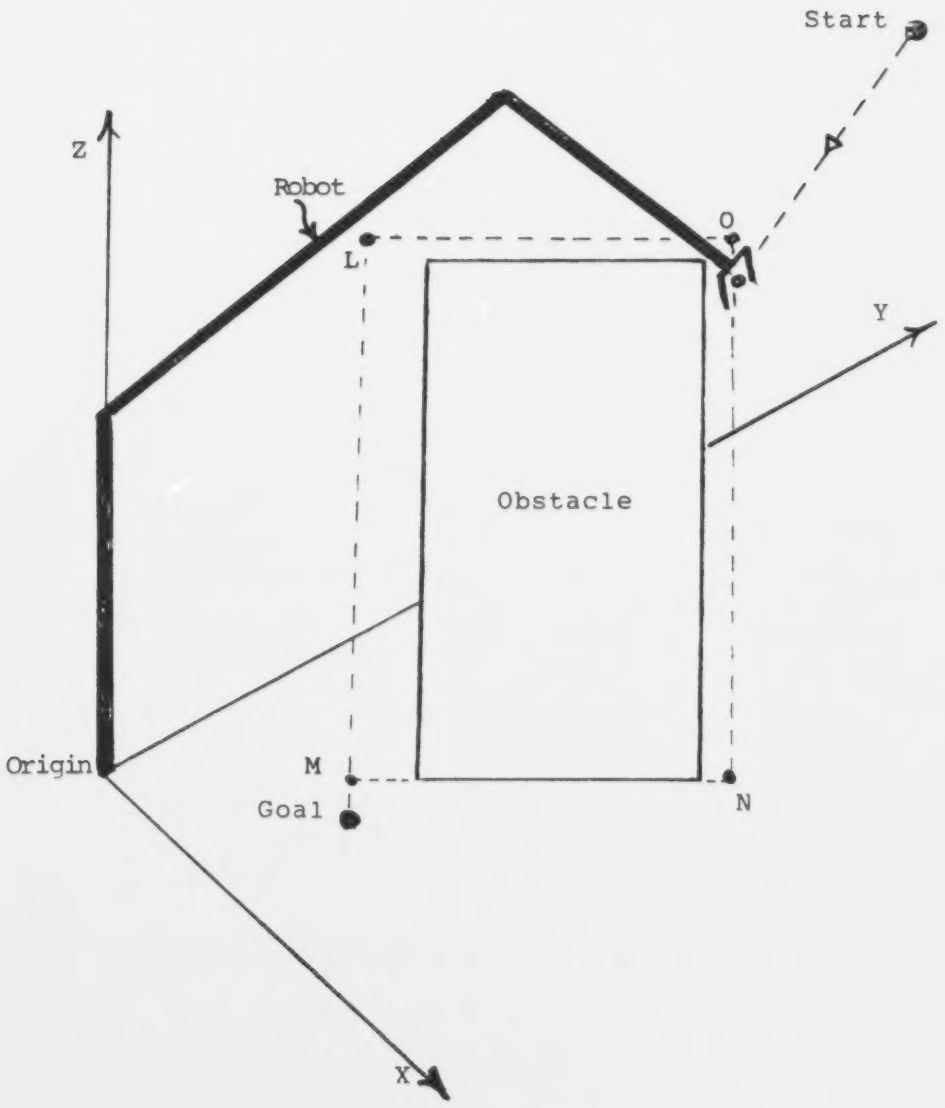


Fig. 4.4: Collision With Gripper Moving Down (Fixed Body Joint Angle).

robot is colliding. They are vertical and in the plane of the robot links. The z values for the top and bottom of the obstacle are known from the geometry data. How the coordinates of important vertices of the rectangles are found is specified below.

4.2.2.1 Gripper Initially Moving Down (Fig. 4.4)

OBSTAP would do as follows:

(a) Carry out the **I**(**O**) computation. Point **O** has coordinates x_o, y_o and z_o such that $x_o = x_{gr}$, $y_o = y_{gr}$ and $z_o = z$ at obstacle top.
Increase z_o if necessary to ensure that links are safely above the obstacle and carry out an updated **I**(**O**) computation.

(b) Carry out the **I**(**L**) computation. Point **L** has coordinates x_L, y_L and z_L such that $x_L = x_g$, $y_L = y_g$ and $z_L = z_o$.

(c) Check for new collisions in moving the gripper to the goal via points **O** and **L**.

(d) IF SAFE, sequentially move the robot or write lines of incremental secondary moves to PATH1.MOT for the following angle changes: $(\theta_{Oi} - \theta_{Pi})$, $(\theta_{Li} - \theta_{Oi})$, $(\theta_{goal,i} - \theta_{Li})$ where $2 \leq i \leq 5$.

(e) IF UNSAFE, an attempt is made to move links as in Section 4.2.1.2 above (Collision With Top/Bottom). The joint angles with the exception of that of the body are corrected to the values they should be at the goal. Finally, the body joint angle is similarly corrected.

4.2.2.2 Gripper Initially Moving Up (Fig. 4.5)

Let us consider a case in which the collision has occurred at the elbow. OBSTAP would do as follows:

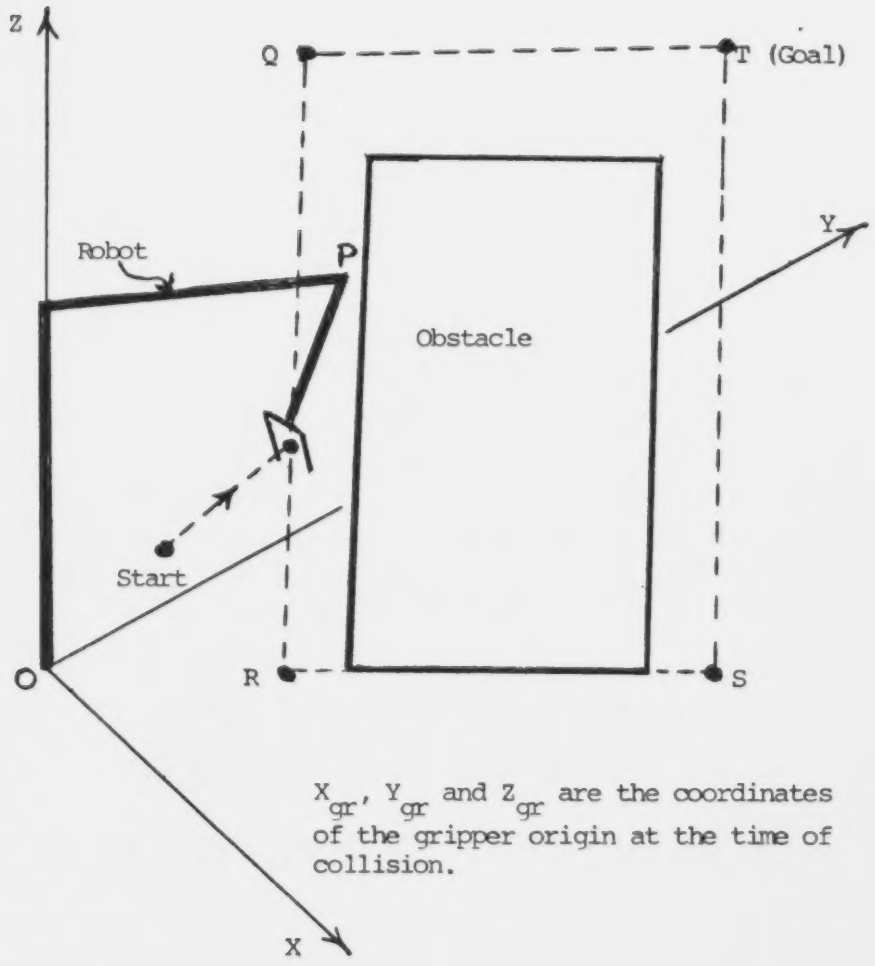


Fig. 4.5: Collision With Gripper Moving up (Fixed Body Joint Angle).

(a) Carry out the **I**(**R**) computation. Point **R** has coordinates x_R, y_R and z_R such that $x_R = x_{gr}$, $y_R = y_{gr}$ and $z_R = z$ at obstacle base.

(b) Carry out the **I**(**Q**) computation. Point **Q** has coordinates x_Q, y_Q and z_Q such that $x_Q = x_R$, $y_Q = y_R$ and $z_Q = z_g$.
Increase z_Q if necessary for cases in which the goal is <u>below</u> the top of the obstacle and carry out an updated **I**(**Q**) computation.

(c) Check for new collisions in moving the gripper to the goal via points **R** and **Q**. (By its not being between the current position and the goal, **R** is an aberrant knot point in this case. It will cause the robot to move away from the goal rather than towards it in the first level path. This point will be cleared out in the second).

(d) IF SAFE, sequentially move the robot or write lines of incremental secondary moves to PATH1.MOT for the following angle changes: $(\theta_{Ri} - \theta_{Pi})$, $(\theta_{Qi} - \theta_{Ri})$, $(\theta_{goal,i} - \theta_{Qi})$ where $2 \leq i \leq 5$.

(e) IF UNSAFE, an attempt is made to move links as in Section 4.2.1.2 above (Collision With Top/Bottom). The joint angles with the exception of that of the body are corrected to the values they should be at the goal. Finally, the body joint angle is similarly corrected.

It should be noted that if the work cell has been moderately cluttered and if the obstacles are dispersed within the space, UNSAFE conditions will not be very common. This will lessen the probability of the occurence of a situation in which the robot will be completely trapped.

4.2.3 Other Factors For First Level

The different steps itemised above are carried out by OBSTAP in planning a path around a detected obstacle. Not all the steps are carried out for every move for which a collision is detected. The relevant steps are determined largely by the kind of move the robot is required to make. If all the possible steps to plan a path have been tried without success the user is alerted to the fact that the robot may be trapped. He is then advised to try using a different growth dimension.

An important consideration has to do with the accuracy of the RM 101. If a computed joint angle is close to the allowed limit for that joint, the robot may crash even if the value of the angle is legal. To reduce the possibility of this happening in OBSTAP, new limits were imposed as follows: body $\pm$110°, shoulder 110° and -20°, elbow -10° and -110°, wrist roll 170° and 10°, wrist pitch $\pm$170°. The result of the imposition of these limits is a reduction in the available work space.

It is possible that the primary move required of the robot may involve the opening or closing of the gripper to help pick up or drop off the payload at knot points. OBSTAP allows for gripper opening/closing only at the end of a move. This contrasts with what happens in an obstacle-free environment in which the gripper would open/close continuously due to joint interpolated control.

4.3 Second Level Path

The main aim of the second level path planning is the reduction in the number of first level knot points and an elimination of unnecessary link movement. Many combinations of knot points may be tested for a collision-free path. The following method, called the Local Path Planning Method, was

employed in coding the program. A Global Method was also employed but only as some support for the Local Method.

Local Path Planning Method

Consider the m knot points of the first level path. which are shown in Fig. 4.6. With the move commencing at point 1 and ending at point m, the path is broken up into segments 1 - 2 - 3, 3 - 4 - 5, 5 - 6 - 7 etc. as far as possible. In the local path planning method, collision checks are carried out for each segment. If, for instance, it is found safe to move the robot from point 1 to 3 directly, then point 2 is eliminated making the number of knot points for the next iteration equal to (m - 1). One iteration is completed when all the segments have been considered. The total number of iterations was set to be 3
after trying different values.

Global Path Planning Method

Fig. 4.7 shows the n knot points for the move after the local path planning has been completed. The move commences at point 1 and ends at point n. The global path planning method is as follows:

(a) Starting from point 1, OBSTAP checks if the straight path from point 1 to point (n - 1) is collision-free.

(b) If the path is collision-free, eliminate knot points 2 to (n - 2) leaving points 1, (n - 1) and n. Then consider the next move required of the robot.

(c) If the path is not collision-free, check if the straight path from point 1 to point (n - 2) is collision-free.

(d) If the new path is collision-free, eliminate knot points 2 to (n - 3) leaving points 1, (n - 2), (n - 1) and n. Then consider the next move required of the robot.

(e) If the new path is not collision-free, check if the

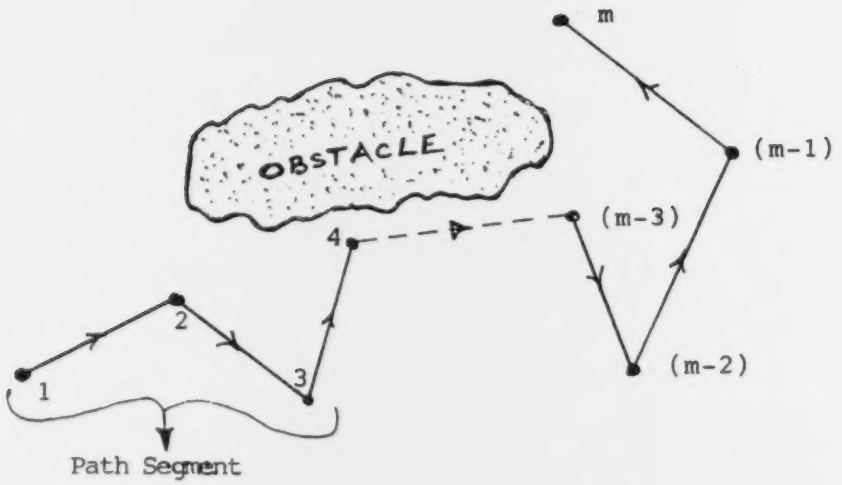


Fig. 4.6: Local Path Planning.

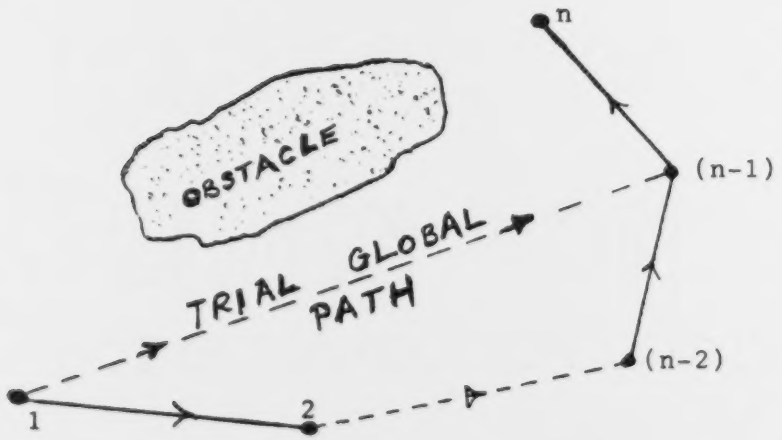


Fig. 4.7: Global Path Planning.

straight path from point 1 to point (n - 3) is collision-free.

(f) Continue the above process until collision checks have been carried out for knot points 1, 2 and 3 with the aim of eliminating point 2.

CHAPTER 5

TEST RESULTS AND DISCUSSION

Program OBSTAP was run in the QUICK BASIC environment on a Gateway 2000 486/33C machine. The results and a discussion of the tests are presented in this chapter.

5.1 Description of Test Obstacles

Five cuboid-shaped styrofoam blocks were first used as obstacles in the testing of the program. These blocks were unconnected with one another and they were scattered within the work cell. The first three of the obstacles were placed on the table top. The other two were elevated above the table so as to be able to observe how OBSTAP would cope with planning a path around obstacles "floating" in the work space of the revolute manipulator (Fig. 5.1).

Two complex obstacles were also considered. This was done so as to find out how OBSTAP would cope with cases of multiple collisions. The complex obstacles consisted of attached simple obstacles. As shown in Fig 5.2, the first set resembled two attached goal posts. The other resembled a four-legged table (Fig. 5.3). The complexity of the obstacles was due to one or more of the following factors:

(i) the component simple obstacles were attached to one another,

(ii) the components were placed close together,

(iii) the components were placed close to the body of the RM 101.

The dimensions of the ungrown test obstacles and their positions within the work cell of the robot are given in Table 5.1(a,b,c). The unconnected obstacles were not all present at the same time when the tests were being carried out. Rather, what was tested was path planning around

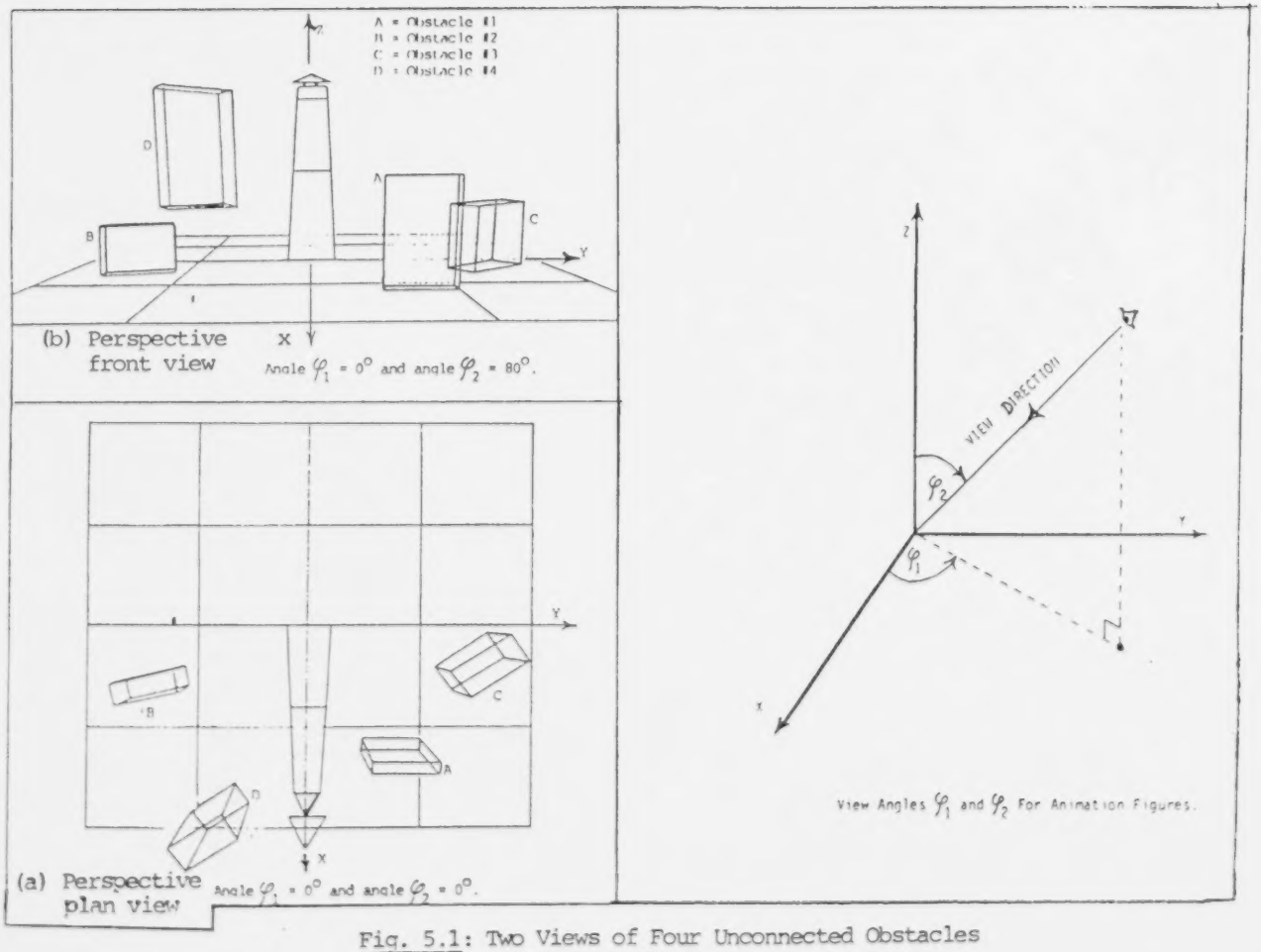

A = Obstacle #1
B = Obstacle #2
C = Obstacle #3
D = Obstacle #4
(b) Perspective front view
Angle φ_1 = 0° and angle φ_2 = 80°.
(a) Perspective plan view
Angle φ_1 = 0° and angle φ_2 = 0°.
VIEW DIRECTION
View Angles φ_1 and φ_2 For Animation Figures.

Fig. 5.1: Two Views of Four Unconnected Obstacles (File OB4.DAT).

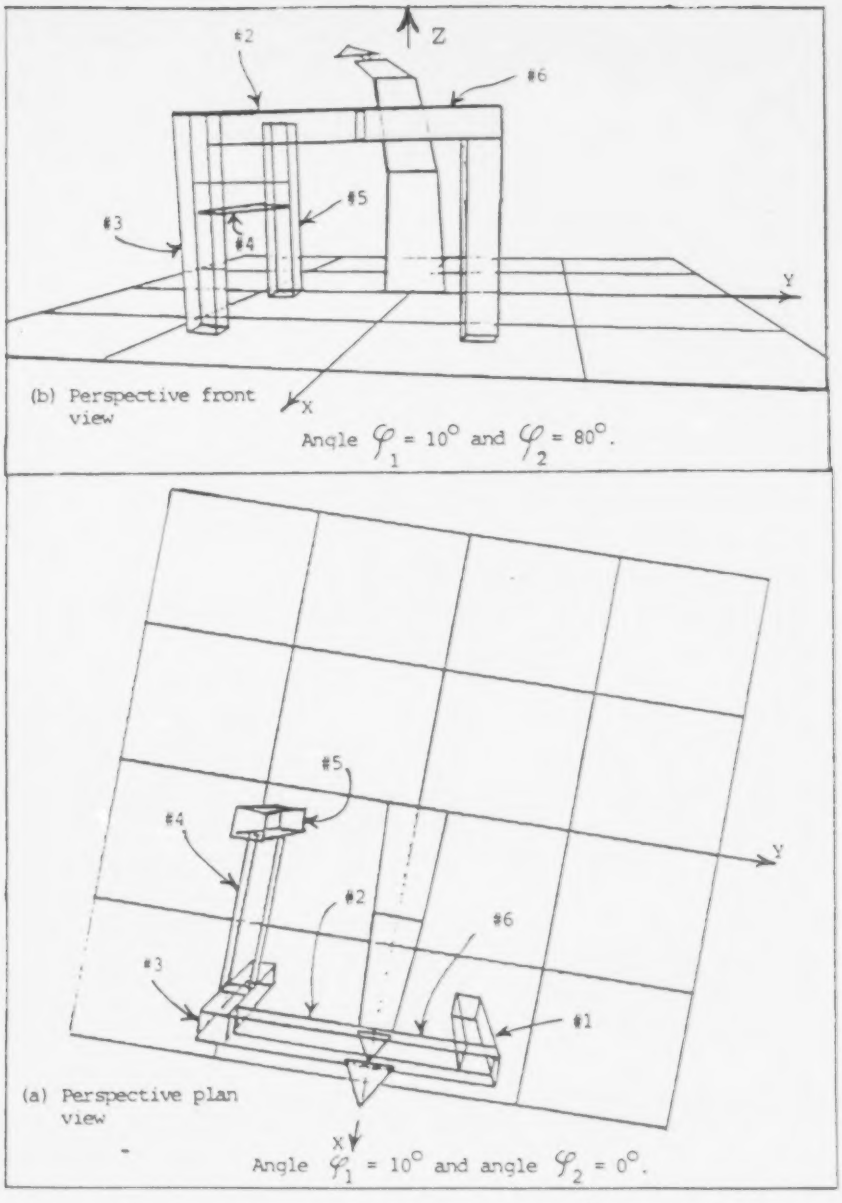


Fig. 5.2: Two Views of Six Connected Obstacles (Set #1, File COB1.DAT).

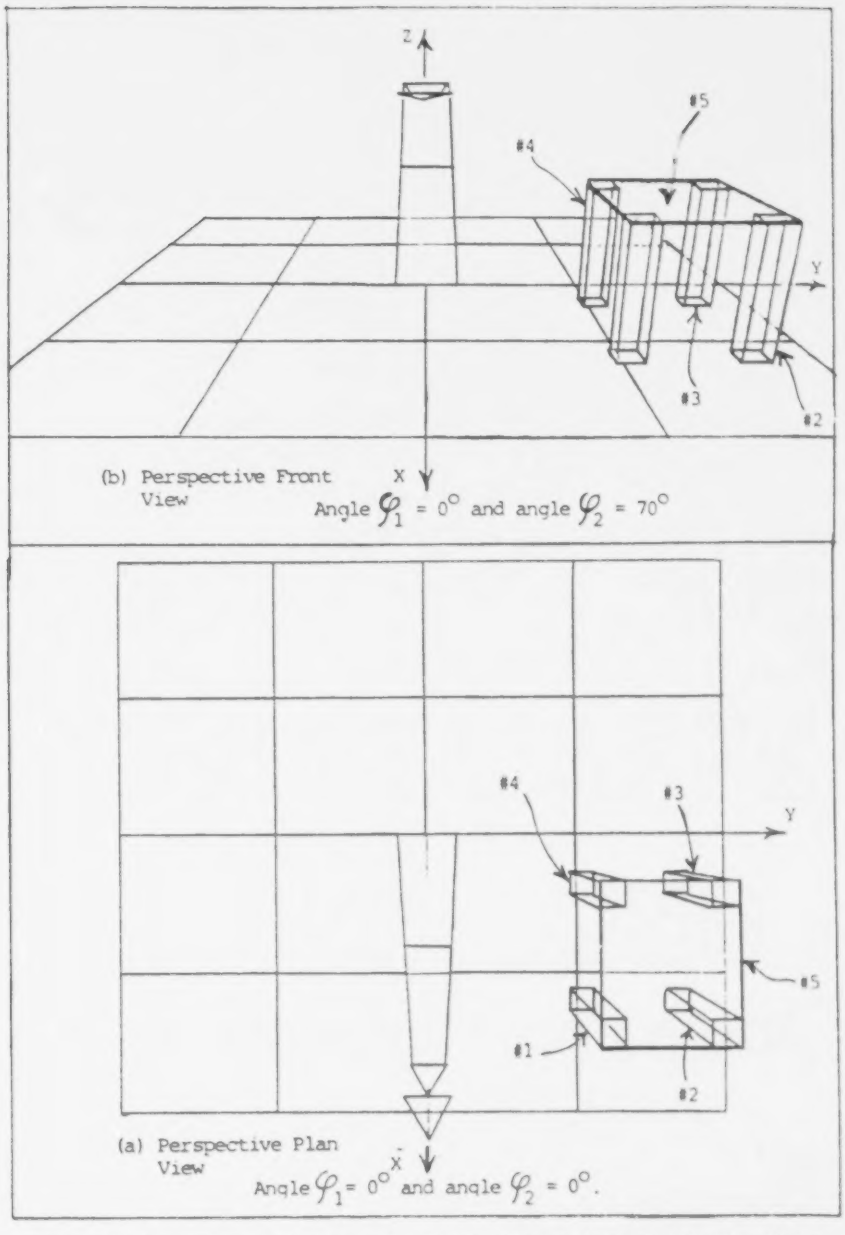


Fig. 5.3: Two Views of Five Connected Obstacles (Set #2, File COB2.DAT).

Table 5.1(a): Dimensions and Locations of Ungrown and Unconnected Obstacles.

Obstacle No.	Dimensions (mm) [Length] [Width] [Height]	Point #1 Location in mm. [x] [y] [z]	Point #2 Location in mm. [x] [y] [z]	Point #3 Location in mm. [x] [y] [z]
1	126	280	306	280
	26	120	120	246
	203	0	0	0
2	136	100	151	135
	52	-280	-270	-414
	104	0	0	0
3	140	100	158	22
	71	280	320	397
	140	0	0	0
4	104	340	382	398
	50	-140	-113	-225
	180	157	157	157
5	104	286	437	344
	180	-176	-76	-264
	50	157	157	157

- File Names and Descriptions (Extension .DAT)

OBST1 (Obstacle 1 only); OB2 (Obstacles 1 and 2);
OB3 (Obstacles 1, 2 and 3); OB4 (Obstacles 1, 2, 3 and 4);
OB13 (Obstacles 1 and 3); OB24 (Obstacles 2 and 4);
OB25 (Obstacles 2 and 5).

Table 5.1(b): Dimensions and Locations of Ungrown Connected Obstacles (First Set).

Obstacle No. In The Set COB1.DAT	Dimensions (mm) [Length] [Width] [Height]	Point #1 Location in mm. [x] [y] [z]	Point #2 Location in mm. [x] [y] [z]	Point #3 Location in mm. [x] [y] [z]
1	40	320	360	320
	40	120	120	160
	250	0	0	0
2	180	320	360	320
	40	0	0	-180
	40	250	250	250
3	40	320	360	320
	40	-180	-180	-220
	290	0	0	0
4	240	80	320	80
	40	-180	-180	-220
	40	160	160	160
5	40	40	80	40
	40	-180	-180	-220
	300	0	0	0
6	160	320	360	320
	40	0	0	160
	40	250	250	250

File Name Description (Extension .DAT)

COB126 Three components from COB1.DAT
Previous # in COB1.DAT (1,2,6)
Respective New # in COB126.DAT (1,2,3)

COB345 Three components from COB1.DAT
Previous # in COB1.DAT (3,4,5)
Respective New # in COB345.DAT (1,2,3)

Table 5.1(c): Dimensions and Locations of Ungrown Connected Obstacles (Second Set).

Obstacle No. In The Set COB2.DAT	Dimensions (mm) [Length] [Width] [Height]	Point #1 Location in mm. [x] [y] [z]	Point #2 Location in mm. [x] [y] [z]	Point #3 Location in mm. [x] [y] [z]
1	40	280	320	280
	40	240	240	280
	200	0	0	0
2	40	280	320	280
	40	400	400	440
	200	0	0	0
3	40	70	110	70
	40	400	400	440
	200	0	0	0
4	40	70	110	70
	40	240	240	280
	200	0	0	0
5	250	70	320	70
	200	240	240	440
	4	200	200	200

File Name Description (Extension .DAT)

COB22 Only the first four components in COB2.DAT.
(identification number of an obstacle being the)
(same in both files)

NCOB2 All five component obstacles in COB2.DAT were used after being re-ordered in their file.
Previous # in COB2.DAT (1,2,3,4,5).
Respective new # in NCOB2.DAT (1,4,5,3,2).

different combinations of the obstacles. For some tests involving the complex obstacles, all the component simple obstacles in a given set were present. For some other tests, some of the component obstacles in the set were removed or the set was rearranged in the work cell. The obstacle combinations used are given as footnotes below the table.

5.2 Description of Test Tasks

For the definition of the task required of the robot, the user of OBSTAP has three options based on inverse kinematics (type A), joint angles (type B) and a motor file (type C). Twelve different task files were compiled to test all three options. Using various combinations of tasks and obstacles eighteen tests were carried out. The default growth dimension of 50 mm was used for most of the tests. There was one type A test, five type B tests and twelve type C tests.

Table 5.2 gives the following details on the task files: the option type on which each file was based, the number of primary moves in the original file and the time that the robot takes to carry out all the moves in the file in the absence of the colliding obstacles. This time is termed execution time. It should be noted that executing a task file is possible only if it is actually a motor file.

5.3 Test Results

These are detailed in Tables 5.3 - 5.5 and Figures 5.4 - 5.6. For each test Table 5.3 gives information on:

(a) the task file,
(b) the corresponding obstacles,
(c) the growth dimension used,
(d) the time taken for OBSTAP to plan a path using the option in which the first level path is observed step by step,

(e) the CPU time required for the completion of computations for PATH1.MOT using the option for motor file compilation,
(f) the CPU time required for the completion of computations for both PATH1.MOT and PATH2.MOT using the option for motor file compilation.

Table 5.4 gives information on:
(a) the time required to execute PATH1.MOT,
(b) the time required to execute PATH2.MOT,
(c) the number of secondary moves in PATH1.MOT for cases of full compilation,
(d) the number of secondary moves in PATH2.MOT also for cases of full compilation.

Five ratios have been defined in order to compare (a) the second level path and the original task and (b) the second level path and the first level. The values of these ratios for each test are given in Table 5.5 and are discussed in Section 5.4. Some comments are also indicated in the table for the complex obstacle tests.

Figures 5.4 - 5.6 show animations for test number 8 (task file TEST12.MOT and obstacle file OB2.DAT). The following are depicted:
(a) the original task file and the test obstacles (Fig. 5.4),
(b) the planned first level path (Fig. 5.5),
(c) the planned second level path (Fig. 5.6).

Table 5.2: Data For Robot Task Files.

No.	Original Robot Task File	Option Type	No. of Primary Moves	Execution Time (sec)
1	TASK1.DAT	A	9	N/A
2	MOVE1.DAT	B	3	N/A
3	MOVE2.DAT	B	8	N/A
4	MOVE3.DAT	B	3	N/A
5	MOTION1.MOT	C	16	136.0
6	MOTION2.MOT	C	7	28.1
7	MOTION3.MOT	C	12	55.4
8	MOTION4.MOT	C	8	36.3
9	TEST1.MOT	C	3	15.6
10	TEST12.MOT	C	8	56.2
11	TEST2.MOT	C	5	20.5
12	TEST3.MOT	C	6	26.4

N/A = Not Applicable. Execution times can be determined only for motor files.

Table 5.3: Test Results (1).

No.	Task File	Obsta-cles (Exten-) (sion) (.DAT)	Growth Dim. (mm)	1st Level Time (sec)	PATH1 CPU Time (sec)	PATH 1&2 CPU Time (sec)
1	TASK1 .DAT	OB13	50	112.1	11.9	38.8
2	MOVE1 .DAT	OBST1	50	43.1	3.7	15.3
3	MOVE2 .DAT	OB24	50	71.2	6.4	29.2
4	MOTION1 .MOT	OB2	50	276.5	22.0	71.9
5	MOTION2 .MOT	OB25	50	100.2	14.8	40.8
6	MOTION3 .MOT	OB4	50	154.3	22.4	67.0
7	TEST1 .MOT	OBST1	50	28.4	2.0	8.0
8	TEST12 .MOT	OB2	50	88.3	6.8	18.5
9	TEST2 .MOT	OBST1	50	34.5	2.4	9.7
10	MOVE3 .DAT	COB1	50	26.7	4.0	
11	MOVE3 .DAT	COB126	40		25.3	154.8
12	MOVE3 .DAT	COB126	50	26.7	3.8	
13	TEST3 .MOT	COB1	40	110.9	17.8	71.5
14	TEST3 .MOT	COB1	50	38.9	6.5	
15	TEST3 .MOT	COB345	50	109.3	16.0	44.2
16	MOTION4 .MOT	COB2	50		5.0	
17	MOTION4 .MOT	COB22	40	106.3	14.1	43.8
18	MOTION4 .MOT	NCOB2	50		15.1	

Table 5.3 gives information on:
(a) the task file,
(b) the corresponding obstacles,
(c) the growth dimension used,
(d) the time taken for OBSTAP to plan a path using the option in which the first level path alone is observed,
(e) the time required for the completion of computations for PATH1.MOT using the option in which motor files are compiled,
(f) the time required for the completion of computations for both PATH1.MOT and PATH2.MOT using the option in which motor files are compiled.

Table 5.4: Test Results (2).

Test Number	PATH1 Execution Time (sec)	PATH2 Execution Time (sec)	No. Of Sec. Moves PATH1	No. Of Sec. Moves PATH2
1	91.7	73.1	68	21
2	37.4	30.7	18	9
3	58.6	51.7	30	19
4	226.4	191.4	99	44
5	82.7	58.5	61	19
6	123.4	80.7	88	19
7	27.1	22.6	16	9
8	74.8	65.2	33	15
9	32.1	26.0	18	6
10	22.8			
11			78	27
12	22.9			
13	90.9	56.0	79	20
14				
15	74.4	53.1	49	18
16	72.7			
17	82.9	63.7	51	22
18				

Table 5.4 gives information on:
(a) the time required to execute PATH1.MOT,
(b) the time required to execute PATH2.MOT,
(c) the number of secondary moves in PATH1.MOT for cases of full compilation,
(d) the number of secondary moves in PATH2.MOT also for cases of full compilation.

Table 5.5: Test Results (3).

Test No.	Ratio R_1	Ratio R_2	Ratio R_3	Ratio R_4	Ratio R_5	Comment
1	N/A	2.33	0.80	0.31	2.26	
2	N/A	3.00	0.82	0.50	3.14	
3	N/A	2.38	0.88	0.63	3.56	
4	1.41	2.75	0.85	0.44	2.27	
5	2.08	2.71	0.71	0.31	1.76	
6	1.46	1.58	0.65	0.22	1.99	
7	1.45	3.00	0.83	0.56	3.00	
8	1.16	1.88	0.87	0.45	1.72	
9	1.27	1.20	0.81	0.33	3.04	
10	N/A	*	*	*	*	C1
11	N/A	9.00	*	0.35	5.12	C2
12	N/A	*	*	*	*	C3
13	2.12	3.33	0.62	0.25	3.02	C4
14	*	*	*	*	*	C5
15	2.01	3.00	0.71	0.37	1.76	
16	*	*	*	*	*	C6
17	1.76	2.75	0.77	0.43	2.11	
18	*	*	*	*	*	C7
Mean	1.64	2.99	0.78	0.40	2.67	

N/A = Not applicable as execution times can be found only for motor files. * = value not computed; see relevant comment.

Table 5.5 gives information on:

(a) ratio R_1 of PATH2.MOT execution time to the execution time of the original type C robot task file,
(b) ratio R_2 of the number of secondary moves in PATH2.MOT to the number of primary moves in the original robot task file,
(c) ratio R_3 of PATH2.MOT execution time to the execution time of PATH1.MOT,
(d) ratio R_4 of the number of secondary moves in PATH2.MOT to the number of secondary moves PATH1.MOT,
(e) ratio R_5 of the CPU time required for second level computations alone to that for the first level computations alone,
(f) any relevant comment which is detailed in Section 5.4.

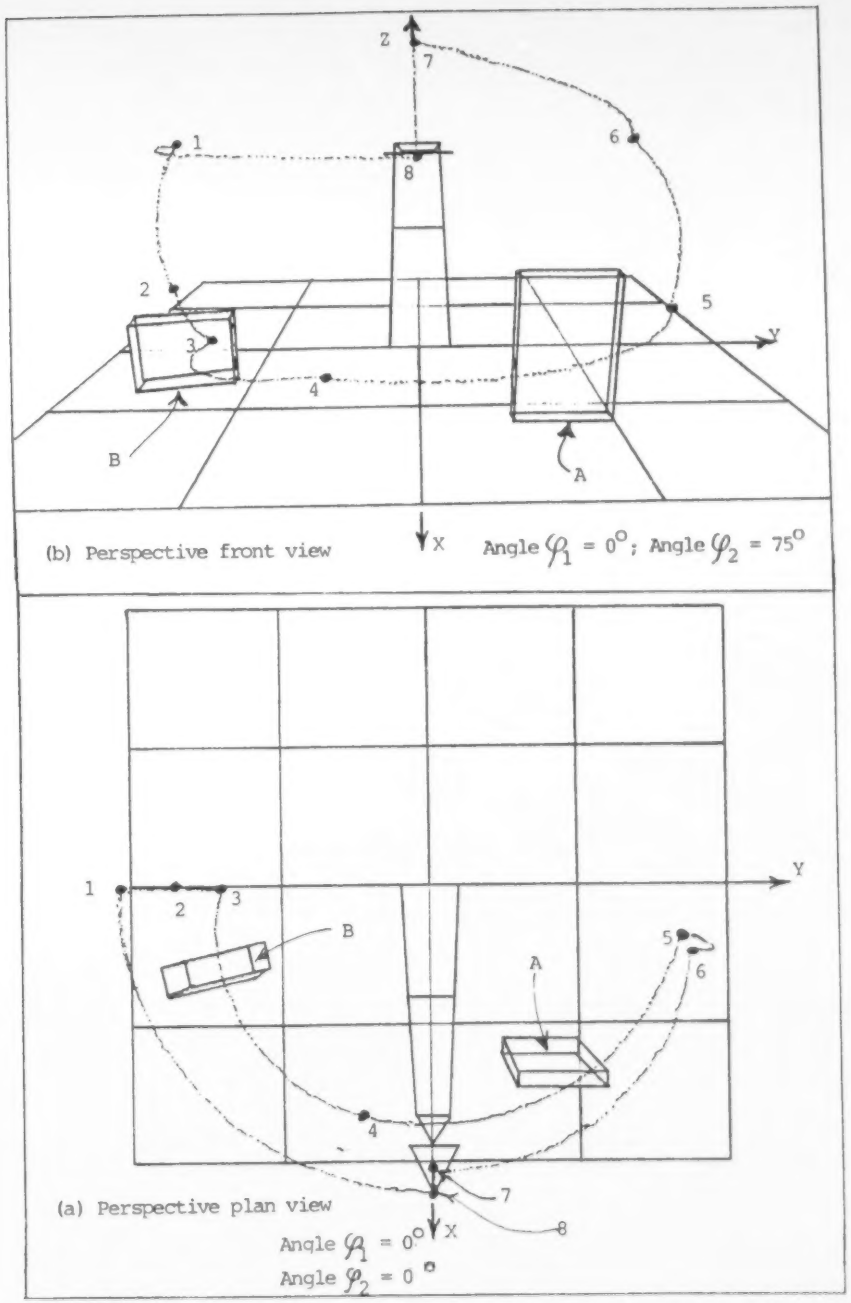


Fig. 5.4: Test #8 Showing Original Motor File (TEST12.MOT) and Two Unconnected Obstacles (OB2.DAT). 1, 2, 3 stand for the goal points of the primary moves. A = Obstacle #1; B = Obstacle #2.

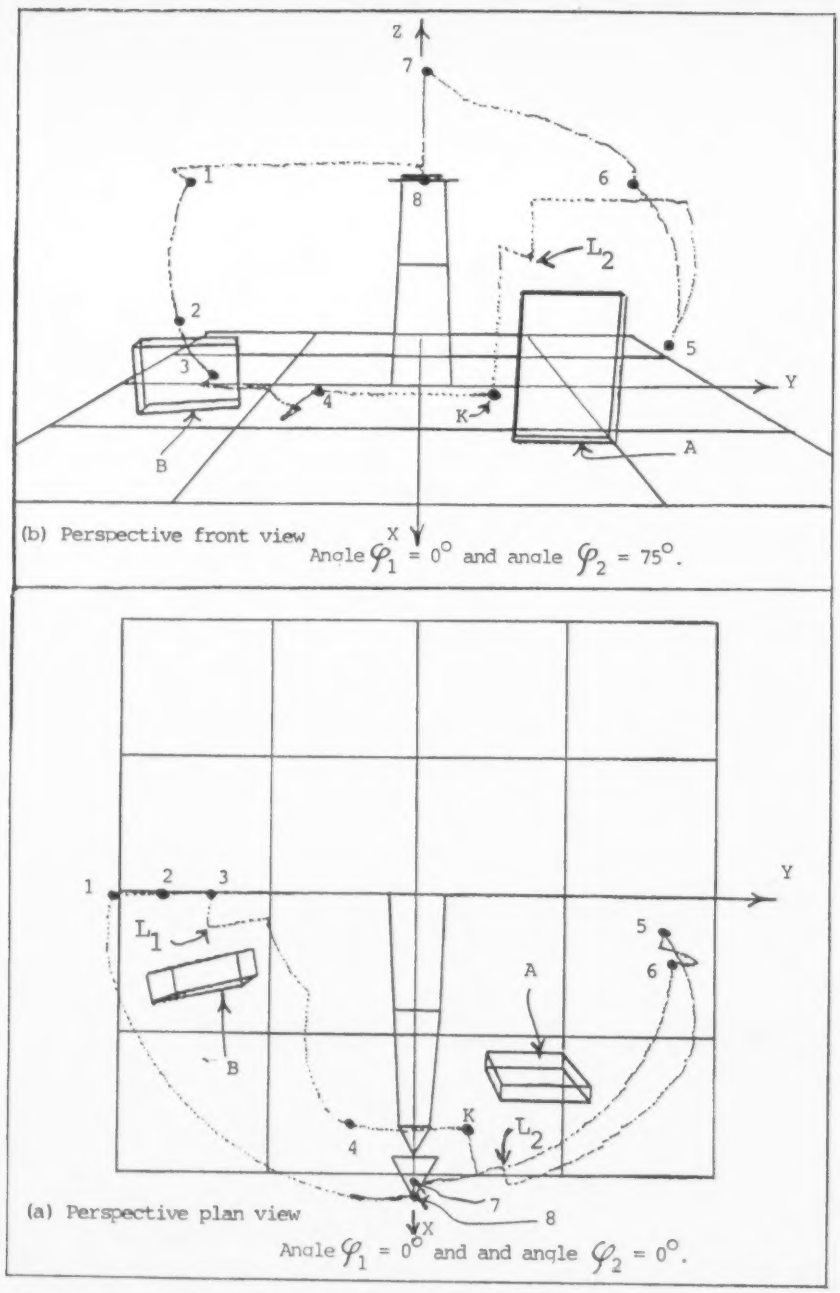


Fig. 5.5: Test #8 Showing the Planned First Level Path Around Obstacles OB2.DAT.

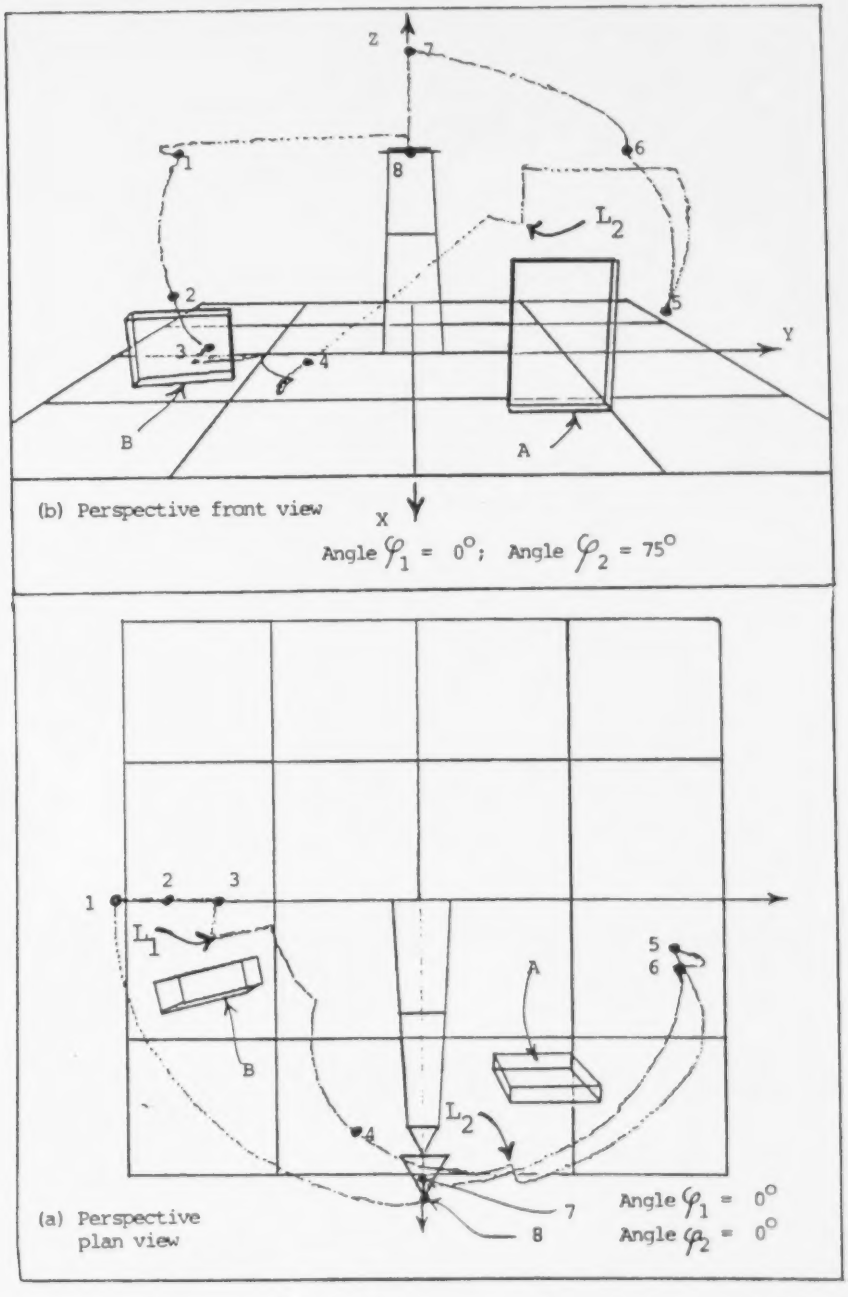


Fig. 5.6: Test #8 Showing the Planned Second Level Path Around Obstacles OB2.DAT.

5.4 Discussion

The results show that OBSTAP can cope excellently with the case of unconnected obstacles (Tests 1 - 9). The program copes fairly well with complex obstacles (Tests 10 - 18). This is supported by the following points:

(a) Program Speed

Test number 6 is taken as an example to discuss the speed at which OBSTAP runs. The test involves twelve primary moves in the original motor file and collisions with all four obstacles placed in the work cell. Despite this, the CPU time is 22.4 sec. for the completion of computations for the first level path and 67.0 sec. for the second. Furthermore, there was no case of the robot being trapped in a local minimum between two or more obstacles.

The speed of program execution was not a crucial factor in this investigation; otherwise a high level computer language would have been used to code OBSTAP instead of QUICK BASIC. The main goal was to develop a search algorithm to find a path past an obstacle. The CPU times noted above would have been smaller still if the program had been written in C or Pascal. The time taken to complete the first level computations is much smaller than the execution time of 55.4 seconds for the original motor file (MOTION3.MOT). The corresponding time for both the first and second levels is not much greater than the execution time.

The above result may be compared with previous similar work on the RM 101. Wong [65] investigated path planning for the robot using a configuration space model written in QUICK BASIC. He considered only rotations at the body, shoulder and elbow joints and treated the forearm and gripper as a single straight link. He reported that for test runs on a 386/33MHz computer, a path search involving two obstacles took approximately one minute for one primary move. A search

involving five obstacles took about 8 minutes [65, p. 71].

(b) Agreement of Results for Menu Options 3 and 4

When observing the first level path directly (Option # 3 in the program menu), the time taken in planning a path around obstacles is approximately equal to the sum of PATH1 CPU time and the PATH1 execution time (Option # 4). This is according to expectation since the first level time involves both computation and movement of robot links in step-wise fashion.

(c) Ratios R_1 and R_2

R_1 is the ratio of PATH2.MOT execution time to the execution time of the original type C robot task file. The ratio is defined only for tests in which the robot task is initially described using a motor file. This is because types A and B task files cannot be downloaded directly to the robot. Ratio R_1 is a measure of the increase in the time required for the completion of a task in the presence of obstacles as against the case of an obstacle-free work cell. A very large value of this ratio would mean much greater production (time) costs for a robot being used in some manufacturing process, even if path planning around some obstacle has been successfully achieved.

It is reasonable to expect that, at best, R_1 will be unity. This is because path planning around an obstacle increases the number of knot points and the travel distance/time of the gripper. From the results table, the mean M_1 for this ratio is 1.64.

R_2 is the ratio of the number of secondary moves in PATH2.MOT to the number of primary moves in the original task file. It is defined for all the task file options. Since path planning around an obstacle must not result in the loss of any original knot point, its ideal value is clearly unity. The mean M_2 for R_2 is 2.99. This value takes into consideration even test 11 for which R_2 = 9.00 (a value

clearly much higher than for the other tests). The value of ratio R_2 for test 11 is extremely high because multiple collisions occurred in the program for the complex obstacle set involved. Neglecting test 11, $M_2 = 2.49$.

It can therefore be stated that by using OBSTAP for path planning around obstacles, the number of secondary moves in the planned path may be more than the number of primary moves in the original task file by an approximate factor of 2.5. The actual value of this factor for a particular test depends on the number of initial collisions, the number of obstacles in the work space and the complexity of the arrangement of the obstacles.

(d) Ratios R_3, R_4 and R_5

R_3 is the ratio of PATH2.MOT execution time to the execution time of PATH1.MOT while R_4 is the ratio of the number of secondary moves in PATH2.MOT to the number of secondary moves in PATH1.MOT. Both ratios R_3 and R_4 described the gain in carrying out path planning at the second level. For a successful second level path, both values must be less than unity. For the tests, the mean (M_3) for ratio R_3 is 0.78. Correspondingly for R_4 we have $M_4 = 0.40$. There is a greater percentage reduction in the number of secondary moves as compared with the percentage reduction in the execution times.

Figures 5.5 and 5.6 may be compared to illustrate the reduction in the number of knot points achieved by carrying out second level path planning. It can be clearly observed that the knot point marked K in the first figure was eliminated in obtaining the second. From Table 5.4 it can also be observed for test #8 that the number of secondary moves is reduced from 33 to 15 by carrying out the second level path planning. However, the kinks at the points marked L1 and L2 in Fig. 5.5 are not removed in Fig. 5.6. They remained even after increasing the number of second level

iterations to 10 (Section 4.3). This may be because the paths shown in the figures are only traces of the gripper. It is possible that the kinks remain because of detected collision with some other link during second level path planning. The presence of the kinks even after more iterations may also indicate that the method used for second level planning is not exhaustive. Probably more combinations of knot points should have been tried with the aim of eliminating some more.

R_s is the ratio of the CPU time required for second level computations alone to that required for the first level alone. With a mean M_s = 2.67, this ratio shows that more time is spent completing the second level computations alone than is required for the completion of the first level computations alone. Again excluding the aberrant ratio value of 5.12 for test 11, we then have M_s = 2.47.

(e) The Use of a Growth Factor

The method of shrinking robot links and correspondingly growing obstacles to have a simplified world model has been proved possible for a revolute robot. The dimension by which obstacles are grown may be changed from its default value of 50 mm by the program user. This default value was fixed based on the largest gripper opening. This dimension or some other choice by the user is the single value by which all the links are shrunk. It should be borne in mind that the cross-section varies along the lengths of the links. Also for a revolute manipulator, there is normally relative rotation between a particular link and some object in the work cell during a move. This means the use of a single parameter for the shrinking of the links is a great simplification of the physical situation. Nevertheless, the results show that this simplification did not prove detrimental.

It was discovered, particularly with unconnected obstacles, that the variability of the growth dimension makes it possible to find more than one planned path for a given

obstacle and primary move. Suppose the retraction of the links past the obstacle has been achieved by using a particular dimension. The use of another dimension may result in the lifting of the links above the obstacle.

The choice of a smaller growth factor may suffice for path planning if the collision is only a slight touch of an obstacle, say by the gripper tip. The program user must not make this factor too small. He has to keep in mind the motor locations on the robot links to prevent collisions at these points (Fig. 2.2). The location of motors 3, 4 and 5 made modelling of the upper arm and the forearm as lines difficult. The problem was whether or not the dimensions of the motors should be included in the value by which the links were shrunk (same as the growth parameter for the obstacles). It was found that including them tended to make the grown obstacles too big (i.e. inefficient use of robot work space) while excluding them meant risking collisions between the motors and obstacles. This may be area for further work. For instance, lines which are perpendicular to the lines which represent the links carrying the motors may be used to model the motors.

A larger growth factor may be used when there is only one obstacle or perhaps two distant obstacles. That is, the possibility of obstacles growing into one another or that of a grown obstacle being too close to the robot body is remote. However, increasing the factor cannot be done indiscriminately. This is because doing so may result in inefficient use of the available work volume and cause the engulfment of either the start point or goal of a primary move by a grown obstacle. Then, although the program user perceives that the robot should be able to reach these two knot points, OBSTAP would always return a collision message.

The need to carefully chose the dimension by which the obstacles are grown therefore raises the following question: Given a particular primary move and a particular obstacle, by what dimension is it best to grow the obstacle? Program

OBSTAP does not answer this question. It is left to the user to supply a reasonable growth dimension. However attempting to answer this question may be worthwhile research. Thus a further improvement to OBSTAP may be that the program be made to automatically select the best growth dimension for a given obstacle and robot move.

(f) Tests With Complex Obstacles

OBSTAP succeeded fairly well in planning paths around the complex obstacles considered. This was despite the fact that the robot was hemmed in by the obstacles which were also placed close together.

Varying the growth dimension was found very useful for this set of tests. For example tests 13 and 14 show that by using a growth dimension of 40 mm, path planning could be fully achieved for task file TEST3.MOT and obstacles COB1.DAT. By using a value of 50 mm for the same run, the obstacles had grown so large that the robot crashed (violation of joint limits).

Table 5.3 gives the PATH1 CPU time for tests 16 and 18 as 5.0 sec. and 15.1 sec. respectively. These were the times that elapsed until the end of the second primary move. The same motor file (MOTION4.MOT) was used for both tests. The same obstacles were also used. However, the order of numbering of the obstacles NCOB2.DAT differed from that used in COB2.DAT. The result shows that the ordering of the obstacles in their file has an influence on the computation times. This is especially true for connected obstacles because the collision routine may need to check for collisions with all the obstacles for a given robot geometry. For disjointed obstacles, some obstacles may be skipped, thus making the influence of the ordering less strong. This, therefore, raises the following question: For a given primary move and set of obstacles, what is the best ordering of the

obstacles in their file? An in-depth look at this may be useful.

Two problems were encountered with the complex obstacles. These were the stack space overflow in QUICK BASIC and the need to plan a path around an undetected obstacle.

Stack space overflow may occur in a QUICK BASIC program if the program has deeply nested or recursive procedures. With the need to repeatedly check for collisions, OBSTAP proves to be such a program. Further, multiple collisions were detected with the connected obstacles. If the overflow occurred for a given primary move and obstacle geometry, it was found that by slightly altering the obstacle geometry (say by using a different growth parameter in another run), the problem might be averted. This is illustrated by the results obtained in tests 11 and 12. For the task file MOVE3.DAT and obstacles COB126.DAT, the use of a growth dimension of 40 mm yielded more results than when 50 mm was used. In the latter situation, overflow was encountered. This is another advantage of the variability of the growth dimension.

OBSTAP was written based on the logic that the robot's planned path will be around the obstacle which has been detected. When obstacles are connected a situation may arise in which a path has to be planned around an obstacle which was not detected. Suppose a given configuration of the robot causes collisions with more than one obstacle. The obstacle around which a path is planned is that which is identified first by the collision routine. This choice may not always be the best and, in part, it explains the limited applicability of OBSTAP to connected obstacles.

Comments

Some details are given in this paragraph on the comments indicated in Table 5.5.

(C1) The data is only for the first primary move in the original robot task file. Stack space overflow occurred

during the second.

(C2) Only CPU times are given; not the time for direct observation of the first level and the motor file execution times. The on-board controller of the RM 101 perceived a violation of the elbow joint upper limit even though to the observer and by calculation such a violation had not yet occurred.

(C3) The data is for the first primary move; stack space overflow occurred during the second.

(C4) This run was successful for an obstacle growth dimension of 40 mm. When 50 mm was tried overflow occurred.

(C5) The data is for the first three primary moves.

(C6) The data is for the first two primary moves and a growth dimension of 50 mm. Overflow occurred when 40 mm was tried.

(C7) The data is for a growth dimension of 50 mm and up to the end of the second primary move. Stack space overflow was encountered when 40 mm was tried.

5.5 Recommendation On Safety

It was observed while testing program OBSTAP that occasionally a link would slightly touch an obstacle followed by a recovery and path planning around the obstacle. The occasional slight touch of an obstacle was observed only for two cases: (i) while observing the first level path directly and (ii) while down-loading the file PATH1.MOT. Such touches were not present for the second level motor file.

Possible reasons for this are as follows:

(i) the RM 101 has positioning accuracy problems,

(ii) the maximum step size of $\pm 1°$ was arbitrarily chosen; it may not be the "best" to use and may need to be reduced especially in the vicinity of an obstacle,

(iii) the permissible minimum distance of 16 mm between the robot and an obstacle may be small.

Furthermore, a length of 85 mm was used for the gripper in the program. This is the distance from the wrist to the middle of the gripper pads (that is the location of the gripper origin). Some touches occurred with the gripper tip. This suggests that for path planning purposes it may be be safer to transfer the gripper axes from where they are normally located all the way to the gripper tip. A length of 100 mm would have been used if this had been done in coding OBSTAP.

A link touching an obstacle should not be considered a big problem. This is because it can be easily rectified by using a larger growth dimension.

It was also observed that the joints could not exactly reach their extreme positions based on the robot specifications. In particular, the elbow joint could not reach close to the published maximum value of 0° without the robot crashing. It was the menu option 3, the direct observation of the first level path, that suffered as a result of this limitation. The crashing of the robot had to be prevented. With option 4, the first level path could recover somewhat from what might have caused a crash in the corresponding option 3 run. The recovery in the second level is more pronounced because of the large reduction in the number of secondary moves associated with the second level path planning.

It is therefore recommended that the observation of a path planned by OBSTAP should be done in the following order:

(i) first viewing an animation of the planned first and second level paths,

(ii) downloading PATH2.MOT to the robot,

(iii) downloading PATH1.MOT to the robot,

(iv) finally, directly observing the first level path.

This order ensures the greatest safety for the robot.

5.6 Application of OBSTAP to Other Robots

The RM 101 for which OBSTAP was written is mainly for instructional purposes. It is smaller than a typical industrial robot which is commonly of the revolute type.

Nevertheless, the principles which have been employed in coding OBSTAP may be applied to other revolute manipulators. Changes will have to be made to the DH parameters. In particular link lengths will have to be increased and joint angle limits will be altered in accordance with the specification of the robot concerned. Codes for the forward and inverse kinematics of the industrial robot will be needed.

The RM 101 has five degrees of freedom (yaw articulation excluded). In contrast, considering a 6-R industrial robot, the gripper will likely be capable of yaw articulation. Therefore, a further modification to OBSTAP will be the use of six joint angles (instead of five as for the RM 101) to monitor robot configuration. Yaw occurs for the gripper alone; since the gripper is already being modelled as a line in OBSTAP, the application of the program to such a industrial robot will not entail the addition of more line models.

If the world origin of an industrial robot is located at the body joint and the axes are positioned as for the RM 101, it is envisaged that the obstacle specification will require no modification. However, there are some robots whose world origins are located at some other points - for example at the shoulder joint. In this case, a modification will also be required because some negative Z coordinates may be permissible. As of now they are not allowed in OBSTAP.

The largest step size has been fixed to be $\pm 1^{\circ}$ in the program. The corresponding minimum distance permissible between any part of the robot and obstacle is 16 mm. A smaller step size may have to be used for an industrial robot and the minimum distance criterion correspondingly modified.

This is because, with the longer links of the industrial robot, its gripper travel when the arm is fully stretched out will be much larger than is found for the RM 101. It must also be noted that for OBSTAP to be applicable, the control of the industrial robot must be joint-interpolated as for the RM 101.

5.7 Limitations of the Heuristic Search

The coding of OBSTAP was based on IF ... THEN statements. The goal of the program was to consider all the different ways in which any part of the RM 101 can collide with an obstacle and to prescribe a safe path for the robot. The test results show that the goal was achieved - particularly for unconnected obstacles for which OBSTAP was written.

However, it must be conceded that there may be some unforseen collision configuration for which the prescribed path planning rules may fail. An important question therefore is: Are the prescribed rules sufficient? The configuration for which such a failure may occur can only be known with the use of the program. The failure may be used to obtain a better version of the program.

CHAPTER 6

CONCLUSION

From the work reported in this thesis, the following can be drawn as a conclusion:

(1) It is possible to use a heuristic world space geometrical approach, such as employed in OBSTAP, to plan paths for a revolute manipulator which operates on joint-interpolated control. With such a method, it is possible to prevent collisions between any part of the robot and fixed obstacles placed in any part of the robot work cell. This is subject to the assumptions made in this thesis on the placement of obstacles and collision types.

(2) The approach used in OBSTAP is fast. For the unconnected obstacles tested, there was no case entrapment of the robot.

(3) The method of shrinking robot links by a single dimension and correspondingly growing obstacles by the same amount has been proved workable for revolute manipulators. This is despite the fact that during a move of such robots there is usually relative rotation between the links and the objects in the work cell. The dimension by which obstacles are grown can be varied by the program user. This variability is particularly useful in finding more than one path between start and goal points. This is helpful in complex obstacle settings.

(4) In quantitative terms, the following were found from test results which lay within a reasonable range:

(i) Ratio R_1 which measured the increase in the time required for the completion of a task in the presence

of obstacles as against the case of an obstacle-free work cell had a mean of 1.64.

(ii) Ratio R_2 which compared the number of secondary moves in the final motor file and the number of primary moves in the original task file had a mean of 2.49.

(iii) Ratio R_3 of PATH2.MOT execution time to the execution time of PATH1.MOT had a mean of 0.78.

(iv) Ratio R_4 of the number of secondary moves in PATH2.MOT to the number of secondary moves PATH1.MOT had a mean of 0.40.

(v) Ratio R_5 had a mean of 2.47, thus showing that more time was spent completing the second level computations alone than was required for the completion of the first level computations alone.

(5) When it was tested the program worked excellently for the kind of obstacles for which it was written (fixed unattached obstacles moderately cluttering the work cell). In all there were nine such tests which were designed to test every subroutine in OBSTAP. Thus it is expected that OBSTAP will be able to handle other similar collision situations. When it was tested the program worked only fairly well with connected obstacles. It therefore needs to be improved for complex obstacle settings. When there is more than one obstacle in the work cell, the order in which the obstacles are numbered in their file has some influence on computation times. Some factors yet to be considered for complex obstacles are the possibility of concurrent multiple collisions and the need to plan a path around an undetected obstacle. Such consideration may help OBSTAP handle complex obstacles more efficiently.

(6) Other problems encountered while testing the program had to do with stack space overflow and robot positioning accuracy. OBSTAP was coded in QUICK BASIC. It is expected that the use of a higher level language will minimise the

overflow problem. It is also expected that the application of a modified version of the program to some other robot which is more accurate than the RM 101 will yield better results.

REFERENCES

1. Harless M and Donath M, "Intelligent Safety System For Unstructured Human/Robot Interaction", Conference Proc. Robotics Int. of SME, Vol 2, 1985.
2. Sugimoto N, "Systematic Robot-Related Accidents And Standardization of Safety Measures For Robots", Proc. 14th Int. Symposium on Industrial Robots, IFS (Publ) Ltd, Kempston, England, 1984, p 131 - 138.
3. Jones R and Dawson S, "People and Robots: Their Safety and Reliability", Proc. 7th British Robot Association Annual Conference, Cambridge, England, May 1984, p 243 - 258.
4. Graham J. H, Meagher J.F and Derby S.J, "Safety and Collision Avoidance System For Industrial Robots", Conference Record - Industry Applications Society, IEEE - IAS Annual Meeting, 1984, p 306 - 313.
5. Kuvin, B.F, "How to Safeguard Welding Robots", Welding Design & Fabrication, Vol 58, 1985, p 72 - 75.
6. Lozano-Perez T and Brooks R.A, "Task-Level Manipulator Programming", Handbook of Industrial Robotics, John Wiley & Sons, 1985, p 404 - 418.
7. Lozano-Perez T and Wesley M.A, "An Algorithm For Planning Collision-Free Paths Among Polyhedral Obstacles", Communications of the ACM, Vol 22, No 10, 1979, p 560 - 570.
8. Wesley, M.A et al., "A Geometric Modelling System for Automated Mechanical Assembly", IBM Journal of Research and Development, Vol 24, No 1, Jan. 1980, p 64 - 74.
9. Brooks R.A, "Symbolic Reasoning Among 3-D and 2-D Images", Artificial Intelligence, Vol 17, 1981, p 285 - 348.

10. Markowsky G and Wesley M.A, "Fleshing Out Wire Frames", IBM Journal of Research and Development, Vol 24, No 5, 1980.

11. Lozano-Perez T, "Automatic Planning of Manipulator Transfer Movements", IEEE Transactions on Systems, Man and Cybernetics, SMC 11, No 10, 1981, p 681 - 689.

12. Ge Q.J and McCarthy J.M, "An Algebraic Formulation of Configuration-Space Obstacles for Spatial Robots", IEEE Int. Conf.on Robotics and Automation, 1990, p 1542 - 1547.

13. Brost R.C, "Computing Metric and Topological Properties of Configuration-Space Obstacles", IEEE Int. Conf. on Robotics and Automation, 1989, p 170 - 176.

14. Warren C.W, Danos J.C and Mooring B.W, "Approach to Manipulator Path Planning", Int. Journal of Robotics Research, Vol 8, No 5, 1989, p 87 - 95.

15. Branicky M.S and Newman W.S, "Rapid Computation of Configuration Space Obstacles", IEEE Int. Conf. On Robotics and Automation, 1990, p 304 - 310.

16. Lozano-Perez T, "Spatial Planning: A Configuration Space Approach", IEEE Transactions on Computers, C32, No. 2, 1983,

17. Deisenroth M.P, "Robot Teaching", Handbook of Industrial Robotics, John Wiley & Sons, 1985, p 352 - 365.

18. Lozano-Perez T, "Robot Programming", Proc. of the IEEE, July 1983,

19. Gruver W.A, Soroka B.I, Craig J.J and Turner T.L, "Evaluation of Commercially Available Robot Programming Languages", 13th Int. Symposium on Industrial Robots and Robots 7, Chicago, April 1983, 12-58 - 12-68.

20. Yong Y.F, Gleave J.A, Green J.L and Bonney M.C, "Off-Line Programming of Robots", Handbook of Industrial Robotics, John Wiley & Sons, 1985, p 366 - 380.

21. Shahinpoor M, A Robot Engineering Textbook, Harper & Row, 1987

22. Lieberman L.I and Wesley M.A, "AUTOPASS: An Automatic Programming System For Computer Controlled Mechanical Assembly", IBM Journal of Research and Development, Vol 21, No 4, 1977, p 321 - 333.

23. Lozano-Perez T, "The Design of a Mechanical Assembly System", M.I.T AI Laboratory, AI TR 397, 1976

24. Popplestone R.J, Amber A.P and Bellos I, "An Interpreter For a Language For Describing Assemblies", Artificial Intelligence, Vol 14, No 1, 1980, p 79 - 107.

25. Borenstein J and Koren Y, "High-Speed Obstacle Avoidance for Mobile Robots", Third Int. Symposium on Intelligent Control, IEEE Systems, Man and Cybernetics Society, 1988, p 382 - 384.

26. Cook K.F and Cipra R.J, "Obstacle Detection and Avoidance Algorithm for a Planar Manipulator", Proc. 15th Design Automation Conference, ASME Design Engineering Division, 1989, p 353 - 360.

27. Warren C.W, "Techniques for Autonomous Underwater Vehicle Route Planning", IEEE Journal of Oceanic Engineering, Vol 15 No.3, 1990, p 199 - 204.

28. Khatib O, Commande dynamique dan's l'espace operationnel des robots manipulateurs en presence d'obstacles, Doctor Ingenieur Thesis, L'Ecole Nationale Superieure de l'Aeronatique et de l'Espace, Toulouse, France, 1980.

29. Warren C.W, "Global Path Planning Using Artificial Potential Fields", Proc. IEEE Int. Conference on Robotics and Automation, Vol 1, 1989, p 316 - 321.

30. Elnagar A and Basu A, "Heuristics for Local Path Planning", Proc. IEEE Int. Conference on Robotics and Automation, Vol 3, May 1992, p 2481 - 2486.

31. Udupa S.M, Collision Detection and Avoidance in Computer Controlled Manipulators, PhD Thesis, Dept. of Electrical Engineering, California Inst. of Tech., 1977.

32. Brooks R.A, "Solving the Find-Path Problem by Representing Free Space as Generalized Cones", M.I.T AI Laboratory, AI Memo 674, 1982.

33. Mehrotra R and Krause D.M, "Obstacle-Free Path Planning for Mobile Robots", 3rd Int. Conf.on Image Processing and Its Applications, IEE Conf. Publication, no 307, 1989,p 431 - 435.

34. Faverjon B, "Obstacle Avoidance Using An Octree in the Configuration Space of a Manipulator", IEEE Int. Conf.on Robotics, 1984, p 504 - 512.

35. Noborio H, Fukuda S and Arimoto S, "Fast Algorithm for Building the Octree for a Three-Dimensional Object From its Multiple Images", Proc. 9th Int. Conf. on Pattern Recognition, IEEE, 1988, p 860 - 862.

36. Kampmann P and Schmidt G, "Topologically Structured Geometric Knowledge Base and Global Trajectory Planning for The Autonomous Mobile Robot MACROBE", Roboter Systeme, Vol 5, No 3, 1989, p 149 - 160.

37. Shin Y.S and Bien Z, "Novel Method of Collision-Free Trajectory Planning for Two Robot Arms", Proc. IEEE Int. Conference on Systems, Man and Cybernetics, Vol 2, 1988, p 791 - 794.

38. Choi Y.J, Crane C.D, Matthew G.K and Duffy J, "Geometry of Interference With Application to Obstacle Avoidance", Proc. 21st Biennial Mechanism Conference, ASME Design Engineering Division, Vol 24, 1990, p 327 - 336.

39. Takahashi O and Schilling R.J, "Motion Planning in a Plane Using Generalized Voronoi Diagrams", IEEE Trans. on Robotics and Automation, Vol 5 No 2, 1989, p 143 - 150.

40. Evans J, Krishnamurthy B, Pong W, Croston R, Weiman C and Engelberger G, "Helpmate - A Robotic Materials Transport System", Robotics, Vol 5 No.3, 1989, p 251 - 256.

41. Soldo M.H, "Fusion Without Representation", Proc. of SPIE: Int. Society for Optical Engineering, Vol 1198, 1989, p 513 - 519.

42. Ganesh C, Dietz G and Jambor J, "Ultrasonic Sensor-Based Motion Control for Robotic Manipulators", Proc. IEEE Int. Conference on Systems, Man and Cybernetics, Vol 2, 1989, p 796 - 797.

43. Wegerif D.G and Rosinski D.J, "Sensor-based Whole-arm Obstacle Avoidance for Kinematically Redundant Robots", Proc. International Society for Optical Engineering, Vol 1828, 1993, p 417 - 426.

44. Shiller Z and Dubowsky S, "Robot Path Planning With Obstacles, Actuator, Gripper and Payload Constraints", Int. Journal of Robotics Research, Vol 8, 1989, p 3 - 18.

45. Chien Y.P, "An Algorithmic Approach to the Trajectory Planning for Multiple Robots", Proc. 22nd Southeastern Symposium on System Theory, 1990, p 303 - 307.

46. De Luca A, Lanari L, Oriola G and Nicolo F, "Sensitivity Approach to Optimal Spline Robot Trajectories", Proc. of the 2nd IFAC Symposium, Series #10, Pergamon Press, 1989, p 505 - 510.

47. Cao Y, Collision-Free Motion Planning and Application to the PUMA 560 Manipulator, M.Sc Thesis, University of Alberta, 1990.

48. Carrioli L and Diani M, "New Algorithm for the Shortest Path Search", Alta - Frequenza, Vol 58, IAN - CNR, Italy, 1989, p 287 - 291.

49. Xue Q and Sheu P, "Path Planning for Two Cooperating Robot Manipulators", Proc. IEEE Int. Workshop on Tools for Artificial Intelligence, Oct. 1989, p 649 - 657.

50. Mayorga R.V, Wong A.K.C and Ma K.S, "Efficient Local Approach for the Path Generation of Robot Manipulators", Journal of Robotic Systems, Vol 7, No 1, 1990, p 23 - 55.

51. Chou L and Song S, "Geometric Work of Manipulators and Path Planning Based on Minimum Energy Consumption", Proc. 21st Biennial Mechanisim Conference, ASME Design Engineering Division, Vol 24, 1990, p 319 - 326.

52. Sandgren E and Venkataraman S, "Robot Path Planning and Obstacle Avoidance - A Design Optimization Approach", Proc. 15th Design Automation Conference, ASME Design Engineering Division, Vol 2, 1989, p 169 - 175.

53. Kant K and Zucker S.W, "Toward Efficient Trajectory Planning: The Path-Velocity Decomposition", Int. Journal of Robotics Research, Vol 5 No. 3, 1986, p 72 - 89.

54. Wu C and Jou C, "Study on the Movement Capability of Robot Manipulators", Proc. 28th IEEE Conference on Decision and Control / Symposium on Adaptive Processes, Vol 3, 1989, p 2500 - 2505.

55. Gourdeau R and Schwartz H.M, "Optimal Control of a Robot Manipulator Using a Weighted Time-Energy Cost Function", Proc. IEEE Conference on Decision and Control / Symposium on Adaptive Processes, Vol 2, 1989, p 1628 - 1631.

56. Han J and Shi Z, "Robot Planning by Analogy", Proc. 1988 IEEE Int. Conference on Systems, Man and Cybernetics, Vol 2, p 781 - 782.

57. Dupont P.E and Derby S, "Simple Heuristic Path Planner for Redundant Robots", Proc. 20th Biennial Mechanism Conference, ASME Design Engineering Div., 1988, p 429 - 440.

58. Ghosh A and Patrikar A.M, "Optimum Path Planning Using the Method of Local Variations", SME Technical Paper, MS1989, p 280 ff.

59. Muthuswamy S and Manoochehri S, "Optical Dynamic Path Planning for Robot Manipulators", Proc. 20th Biennial Mechanisms Conference, ASME Design Engineering Div., 1988, p 517 - 524.

60. Martin D.P, Baillieul J and Hollerbach J.M, "Resolution of Kinematic Redundancy Using Optimization Techniques", IEEE Transactions on Robotics and Automation, Vol 5, 1989, p 529 - 533.

61. Paul R.P, Robot Manipulators: Mathematics, Programming and Contol, MIT Press, 1981.

62. Instruction Manual For the RM 101 Micro-Robot, Mitsubishi Electric Corporation.

63. Toogood R.W, "A Work Cell Animator for Robotics Instruction", Computers In Engineering, ASME, Vol 2, 1991, p 361 ff.

64. Parkin R.E, Applied Robotic Analysis, Prentice-Hall, 1991.

65. Wong C, Robot Path Planning and Obstacle Avoidance in Configuration Space, M.Eng Thesis, University of Alberta, 1993.

APPENDIX: OBSTAP USER'S MANUAL

Some details which could not be included in the second chapter of the thesis are now given. For the sake of completeness some ideas which were stated in that chapter are repeated here.

A1. Uses of OBSTAP

The program OBSTAP has about 8,100 lines of code. It requires some 283 kbytes of memory and is run in the QUICK BASIC environment. Thus when OBSTAP is to be run on small RAM machines few obstacles and robot moves should be considered so as to prevent memory problems.

OBSTAP can be used for several purposes. It can be run to enter data which describes either the moves to be made by the robot or the obstacles in the work cell. Any data entered for the first time is saved so that it need not be entered from the the keyboard for subsequent runs. OBSTAP can also be run in order to carry out obstacle detection with or without path planning around the detected obstacle. The user also has the option of compiling the obstacle data being entered for the first time for animation purposes.

In order to input robot task data alone, the user has to enter 1 when prompted by the menu screen. To input obstacle data alone, 2 has to be entered. Entering 3 causes OBSTAP to detect obstacles without path planning while by entering 4 both obstacle detection and path planning can be done. Thereafter the user should simply follow instructions printed to the screen for the choice he has made.

A2. Description of Robot Task

In specifying a task for the RM 101, it is assumed that the robot is initially in its standard home position [62]. A number of distinct moves constitute a task. OBSTAP

has been coded to handle up to 20 moves for a given task. However, because of possible computer memory limitations, it is recommended that a task should consist of only a few moves at a time. The last move for a given task should end at the home position.

There are three ways in which the user can describe a move. These are as follows:

(A) To enter the x, y, z coordinates and the orientation of the gripper at the goal knot point of all the moves except the final move.

(B) To enter the joint angles at the end of all the moves except the final move.

(C) Including the move in a motor file of incremental moves which is compiled as specified in the robot manual.

It can be seen that choice A is based on the inverse kinematic approach. Choice B is based on forward kinematics. Choice C is also based on forward kinematics, this time using step changes in joint angles. When carrying out obstacle detection (with or without path planning) data using options A or B may be entered from the keyboard or read from files. Data entered from the keyboad is saved in the file TASK.DAT in the default drive if the data is based on option A. The data is saved in the file MOVE.DAT if it is based on option B. In the case of option C, the data must be contained in a motor file; it cannot be entered from the keyboard.

It is only with choice C that the opening or closing of the gripper can be specified. It should be noted, however, that the mere opening and closing of the gripper is not recognized by OBSTAP as constituting a move; neither is a mere roll of the gripper recognized. A legal move which necessitates the investigation of obtacle detection must include motion at one or more of the following joints: body, shoulder, elbow and wrist-pitch.

It is recommended that the program user should not make

any knot point too close to the work top. This is to avoid the possibility of the work top itself becoming an obstacle especially for the first level path. All the knot points should be located at least 40 mm above the table.

A3. Description of Obstacles

In OBSTAP, an obstacle is modelled in a simplified form as a cuboid. In practice, a real obstacle may have a different shape. In this case, the program user will have to consider a suitable cuboid which envelops the real obstacle. In order to maximise the use of the robot work space, the volume of the cuboid should be as small as possible.

It is assumed that the work space is moderately cluttered, that is there are at most only a few fixed obstacles present. The maximum number of obstacles allowed has been set at four although, while testing OBSTAP, up to six were tried at a stage. It is also assumed that they are scattered in the work cell and not bunched up in one spot or connected to one another. No part of an obstacle may project below the table top. An obstacle should also be entirely contained in one quadrant. If an object lies in two quadrants, for the purpose of modelling, it should be considered as two obstacles.

The coordinates of three vertices and the height of an obstacle are required to describe the obstacle. The coordinates are taken with respect to the world origin. The plane which contains the vertices is the lower horizontal face of the cuboid. Considering the projection of this face onto the X - Y plane [Fig. A1], vertex 1 is the closest to the origin. Lines 1-2 and 1-3 represent adjacent edges. The obstacle data that is entered from the keyboard is saved in the file OB.DAT in the default drive.

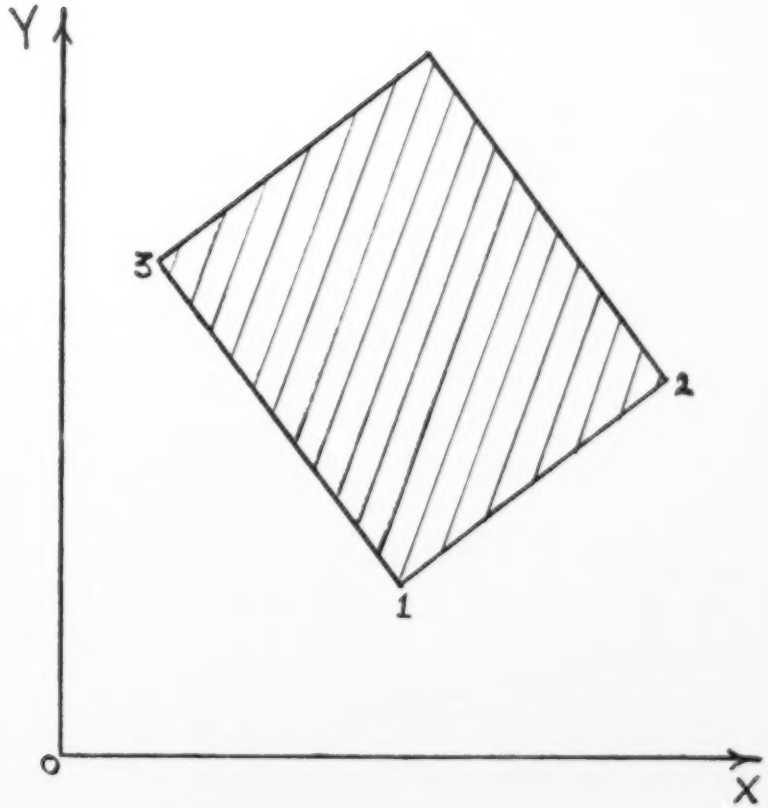


Fig. A1: Obstacle Description in OBSTAP.

There are two options for specifying the orientation of an obstacle relative to the world axes. For a given run involving a set of two or more obstacles, it is not allowed that the data of one obstacle is based on option 1 while that for some other obstacle is based on option 2. Data for all the obstacles in the set must be based on the same option. The two options are:

(1) Having the edge 1-2 parallel to the X axis and the edge 1-3 parallel to the Y axis.

(2) A more general case in which the edge 1-2 is at some acute angle to the X axis; correspondingly, the edge 1-3 is at the same acute angle to the Y axis.

In order to have a simplified model, each link of the robot is shrunk to a line when carrying out either obstacle detection alone or path planning around a detected obstacle. Correspondingly, the obstacles in the work cell have to be grown. The default dimension by which obstacles are grown is 50 mm. However, at a prompt, the user may vary this dimension. It is important to ensure that the desired goal for the robot at the end of a move is not engulfed by or too close to the grown obstacle for the chosen growth dimension.

For a move in which the collision is a slight touch of an obstacle by one of the links, a growth dimension smaller than 50 mm may be tried. For direct impacts the growth dimension may be suitably increased.

A4. Obstacle Detection Only

The user may investigate whether or not a particular move will cause a collision. In this case, OBSTAP does not plan a path around any detected obstacle. The names of the robot task file and the obstacle data have to be supplied. Alternatively, task and obstacle data could be entered from the keyboard. A suitable obstacle growth dimension must also be supplied.

The user has the options of either checking for just one particular move of interest among a set of moves in the task file or checking for all the moves in the file. Detailed collision information is normally printed on the screen. If the user also wants, he may stipulate that hard copies of the information be printed. Any move for which such information is not supplied should be taken as collision-free.

The information is printed only for the first possible collision of a move. It includes the following:

(i) the particular move,
(ii) the obstacle involved,
(iii) the link involved [upper arm, forearm or gripper],
(iv) the side of the obstacle [top, bottom or a vertical side],
(v) all five joint angles at collision,
(vi) the position of the gripper origin relative to the world origin,
(vii) the world coordinates of the collision point.

The collision data is valid only for the chosen growth dimension. It is possible that some other close robot configuration may also cause a collision.

A5. Obstacle Detection With Path Planning

Obstacle detection followed by path planning around a detected obstacle can also be done using OBSTAP. As in Section A4 above, the names of the robot task file and the obstacle data file must be supplied or task and obstacle data must be entered from the keyboard. An obstacle growth dimension must also be supplied.

The user may choose to observe the planned path step by step as it is determined by the program. In this case, the robot must be connected to the computer. Only the first level path can be observed.

Alternatively, the program user may stipulate that planned incremental moves be written to motor files which can be downloaded to the robot at a later stage. For this option, OBSTAP compiles the file PATH1.MOT for the first level path and the file PATH2.MOT for the second level path.

The first level path has many more knot points when compared with the second level path. The first path may show link fluctuation as program OBSTAP tries to find a path for the robot around a detected obstacle. The second level path achieves smoother motion from the start point of a move to the goal by the elimination of some intermediate knot points.

Information is printed to the screen that OBSTAP has found a path for a particular move when computations for that move are completed. The total times required to complete computations for (i) the first level path and (ii) both the first and second level paths are also printed. Only the first level time is printed if the user previously chose to observe the planned path step by step. It should be noted that the robot reaction time is included in this value. Both times (i) and (ii) above are printed if the user previously chose the compilation of motor files. Robot reaction time is not included in this case.

A6. Robot Safety and Path Animation

When the data describing an obstacle is being entered from the keyboard, the program user may specify that animation data should be compiled. No animation routine is contained in OBSTAP itself. However, it can compile work cell data that can be used in the program ANIM written by Toogood [63]. The compiled data is stored in the file WKCELL.DAT.

Animation of the planned path should be done when the user plans to compile motor files PATH1.MOT and PATH2.MOT.

It has the advantage that prior to downloading these files to the robot, the planned paths can be viewed on a monitor. It is recommended that the observation of a path planned by OBSTAP should be done in the following order:

(i) first viewing an animation of the planned first and second level paths,

(ii) downloading PATH2.MOT to the robot,

(iii) downloading PATH1.MOT to the robot,

(iv) finally, directly observing the first level path.

This order ensures the greatest safety for the robot.